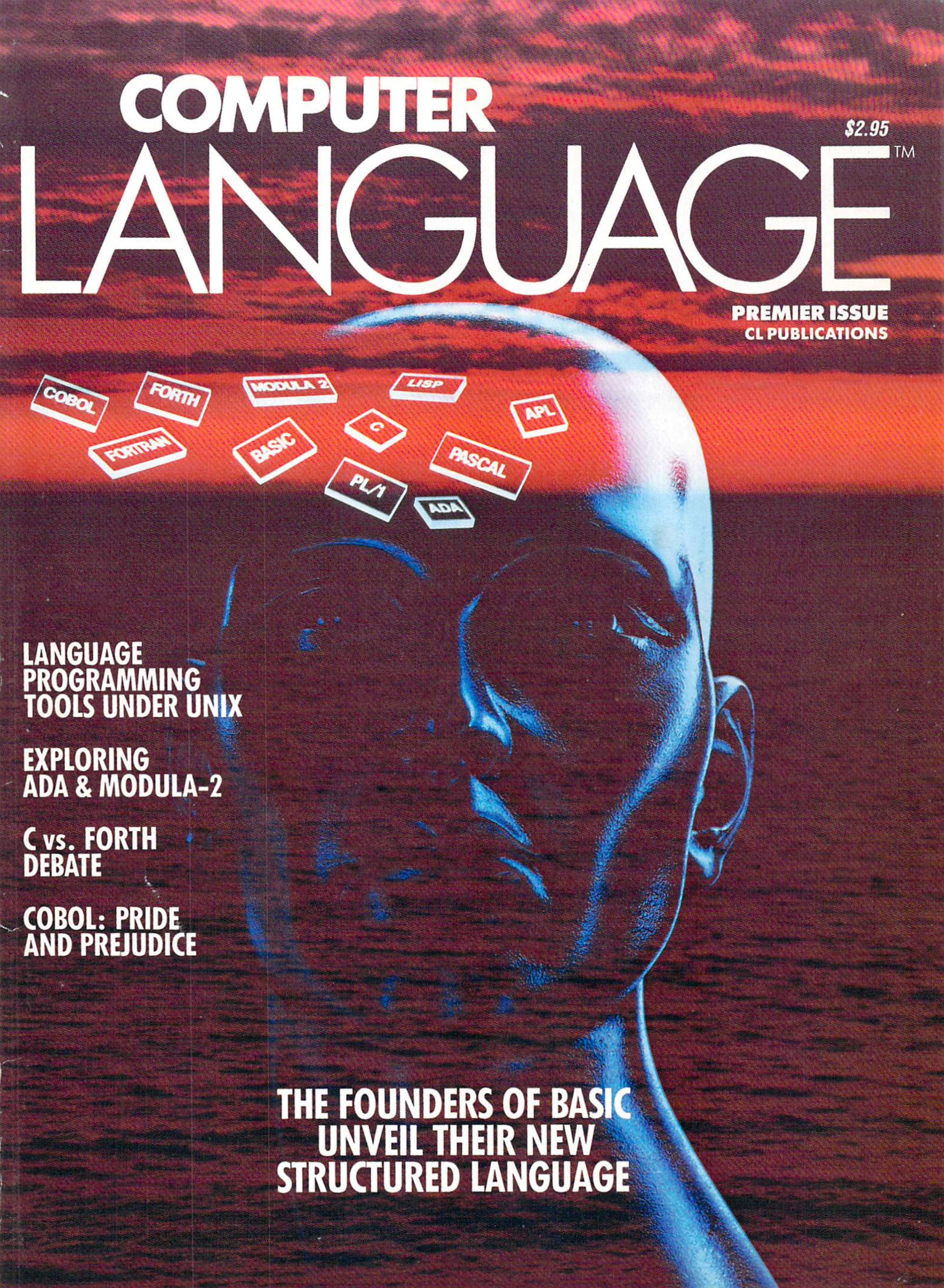
COMPUTER LANGUAGE™

$2.95

PREMIER ISSUE
CL PUBLICATIONS


COBOL
FORTH
MODULA 2
LISP
APL
FORTRAN
BASIC
C
PASCAL
PL/1
ADA

LANGUAGE PROGRAMMING TOOLS UNDER UNIX

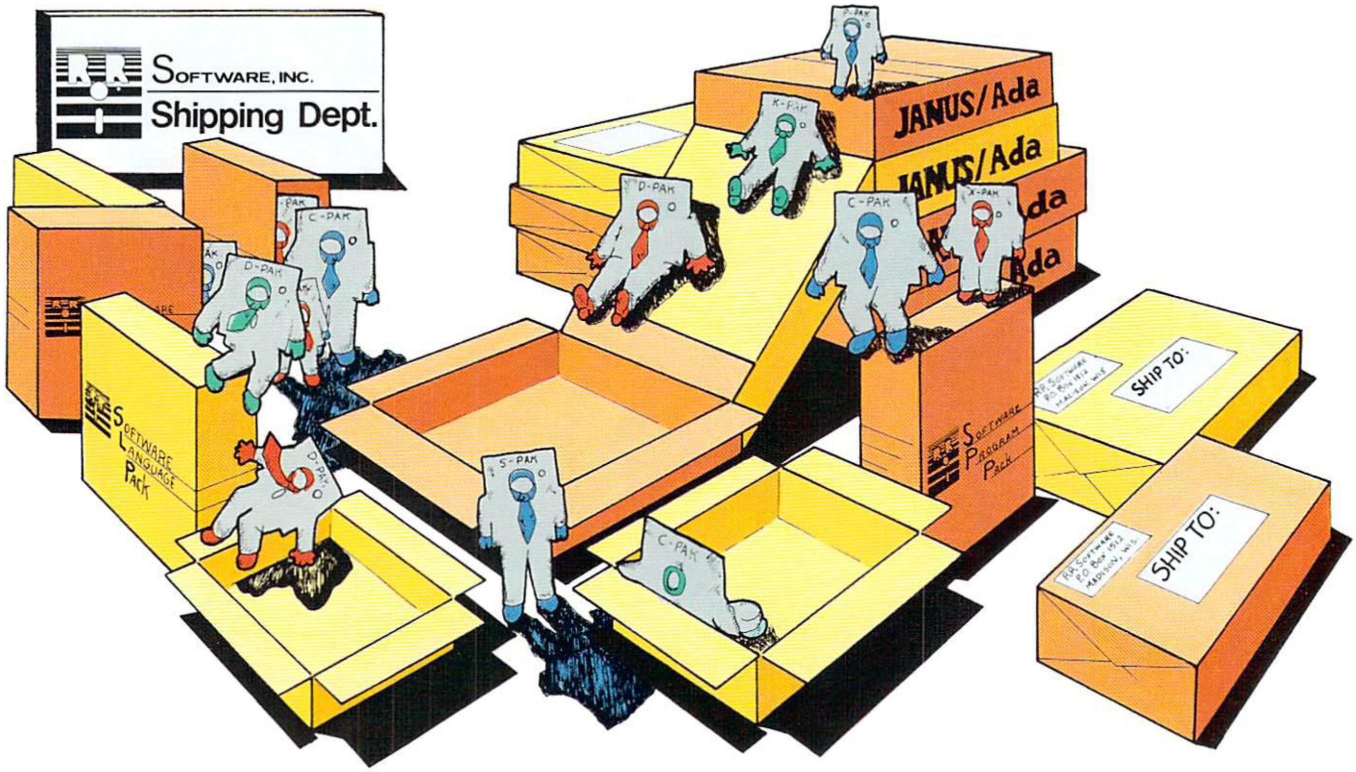
EXPLORING ADA & MODULA-2

C vs. FORTH DEBATE

COBOL: PRIDE AND PREJUDICE

THE FOUNDERS OF BASIC UNVEIL THEIR NEW STRUCTURED LANGUAGE

WE'VE GOT YOUR PACKAGE!!



We offer you the most flexible, cost efficient means of introducing your programming staff to the Ada Language. **You** can choose the level of Support **you** need, when you need it! These Janus/Ada packages are customer-tested and available now...

(C-Pak) Introductory Janus/Ada Compilers
(D-Pak) Intermediate Janus/Ada Systems
(S-Pak) Advanced Janus/Ada Systems
(P-Pak) Janus/Ada Language Translators

Janus/Ada "Site" Licenses
Janus/Ada Source Code Licenses
Janus/Ada Cross Compilers
Janus/Ada Maintenance Agreements

Coming Soon: New Computer and Operating Systems Coverage

Selected Janus/Ada packages are available from the following:

National Distributors

Westico, Inc.
25 Van Zant St.
Norwalk, CT 06855
(203) 853-6880

Soft-Net
5177 Richard, Suite 635
Houston, TX 77056
(713) 933-1828

AOK Computers
816 Easley St., Suite 615
Silver Springs, MD 20910
(310) 588-8446

Trinity Solutions
5340 Thornwood Dr., Suite 102
San Jose, CA 95123
(408) 226-0170

Compuview Products, Inc.
1955 Pauline Blvd., Suite 200
Ann Arbor, MI 48103
(313) 996-1299

International Distributors

Micronix
11 Blackmore St.
Windsor 4030
QLD. Australia
(07) 57 9152

Progesco
155, rue du Fauburg
St. Denis
75010 Paris
(1) 205-39-47

Lifeboat of Japan
S- 13-14, Shiba
Minato-Ku
Tokyo 108 Japan
03-456-4101

CP/M, CP/M-86, CCP/M-86 are trademarks of Digital Research, Inc.
*ADA is a trademark of the U.S. Department of Defense
MS-DOS is a trademark of Microsoft

© Copyright 1983 RR Software

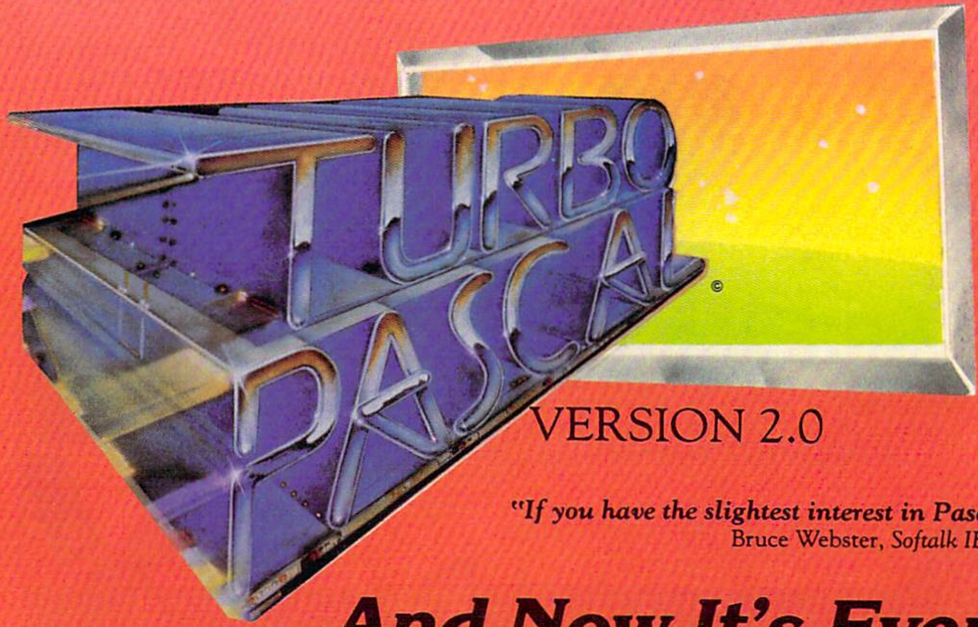
SOFTWARE, INC.

specialists in state of the art programming

P.O. Box 1512 Madison, Wisconsin 53701
(608) 244-6436 TELEX 4998168

CIRCLE 47 ON READER SERVICE CARD

This is THE PASCAL COMPILER You've Been Hearing About



VERSION 2.0

"It's almost certainly better than IBM's Pascal for the PC . . . Recommended."
Jerry Pournelle
Byte, May 1984

$49.95

"If you don't have CP/M [for your Apple], Turbo Pascal is reason enough to buy it."
Cary Hara
Softalk Apple, May 1984

"If you have the slightest interest in Pascal . . . buy it."
Bruce Webster, *Softalk IBM*, March, 1984

And Now It's Even Better Than You've Heard!

- **Windowing (IBM PC, XT, jr. or true compatibles)**
- **Color, Sound and Graphics Support (IBM PC, XT, jr. or true compatibles)**
- **Optional 8087 Support** ***(available at an additional charge)***
- **Automatic Overlays**
- **A Full-Screen Editor that's even better than ever**
- **Full Heap Management—via dispose procedure**
- **Full Support of Operating System Facilities**
- **No license fees.** You can sell the programs you write with Turbo Pascal without extra cost.

Yes. We still include Microcalc . . . the sample spreadsheet written with Turbo Pascal. You can study the source code to learn how a spreadsheet is written . . . it's right on the disk.* And, if you're running Turbo Pascal with the 8087 option, you'll never have seen a spreadsheet calculate this fast before!

**Except Commodore 64 CP/M.*

Order Your Copy of TURBO PASCAL® VERSION 2.0 Today

For VISA and MasterCard orders call toll free: 1-800-227-2400 x968
In California: 1-800-772-2666 x968
(lines open 24 hrs, 7 days a week)

Dealer & Distributor Inquiries Welcome 408-438-8400

C20

Choose One (please add $5.00 for shipping and handling for U.S. orders. Shipped UPS)

- ____ Turbo Pascal 2.0 $49.95 + $5.00
- ____ Turbo Pascal with 8087 support $89.95 + $5.00
- ____ Update (1.0 to 2.0) Must be accompanied by the original master $29.95 + $5.00
- ____ Update (1.0 to 8087) Must be accompanied by the original master $69.95 + $5.00

Check ____ Money Order ____
VISA ____ Master Card ____
Card #: ____________
Exp. date: ____________

BORLAND
INTERNATIONAL

Borland International
4113 Scotts Valley Drive
Scotts Valley, California 95066
TELEX: 172373

CIRCLE 4 ON READER SERVICE CARD

My system is: 8 bit ____ 16 bit ____
Operating System: CP/M 80 ____
CP/M 86 ____ MS DOS ____ PC DOS ____
Computer: ____________
Disk Format: ____________
Please be sure model number & format are correct.
Name: ____________
Address: ____________
City/State/Zip: ____________
Telephone: ____________

California residents add 6% sales tax. Outside U.S.A. add $15.00 (If outside of U.S.A. payment must be by bank draft payable in the U.S. and in U.S. dollars.) Sorry, no C.O.D. or Purchase Orders.

AZTEC

AZTEC C86

Optimized "C" compiler for PC DOS, MS DOS & CP/M-86
PC DOS, UNIX I/O, math, screen, graphics libraries
8086 assembler, linker & librarian, overlays
/PRO — library source, debug, ROM, MASM & RMAC, 8087, large model

NEW C COMPILERS

AZTEC C68K for MACINTOSH
VAX cross compilers

AZTEC C II

Optimized "C" compiler for CP/M, TRSDOS & LDOS
assembler, linker & librarian, overlays, utilities
UNIX I/O, math & compact libraries
/PRO — library source, ROM, M80 & RMAC

C TOOLS & AIDS

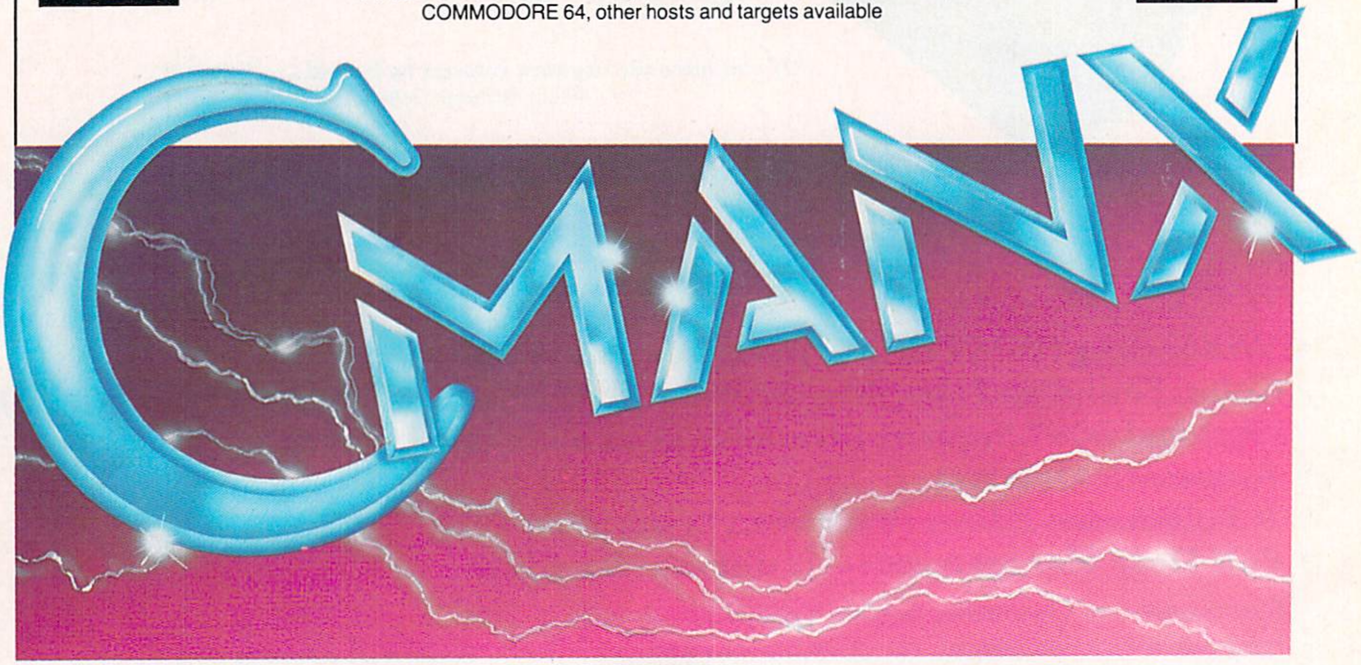
Z editor (like Vi), C TUTOR compiler, PHACT database, C GRAFX, UNI-TOOLS I, QUICK C, BABY BLUE for PC to CP/M cross, QUADLINK for PC to APPLE cross

AZTEC C65

"C" compiler for APPLE DOS 3.3, ProDOS or COMMODORE 64
VED editor, SHELL, UNIX & math libraries
/PRO — library source, ROM, overlays

CROSS COMPILERS

Compile & link on HOST — test on TARGET machine
HOSTS: UNIX, PC DOS, CP/M-86, CP/M-80, VENIX, PCIX, APPLE
TARGETS: PC DOS, CP/M-86, CP/M-80, APPLE, RADIO SHACK, COMMODORE 64, other hosts and targets available



PRICES

AZTEC C86 C COMPILER

PC DOS MSDOS	249
CP/M-86	249
BOTH	399
/PRO EXTENSIONS	249
Z (VI EDITOR)	125
C TUTOR COMPILER	99
PHACT DATABASE	299
C GRAFX	99
SUPERDRAW	299
UNI-TOOLS I	99
QUICK C	125

AZTEC C II C COMPILER

CP/M	199
/PRO EXTENSIONS	150
TRS 80 MODEL 3	149
TRS 80 MODEL 4	199
TRS 80 PRO (3 & 4)	299

AZTEC C65 C COMPILER

APPLE DOS 3.3	199
ProDOS	199
BOTH	299
/PRO EXTENSIONS	99
C TUTOR COMPILER	99
E EDITOR	99
QUICK C	125

AZTEC C CROSS COMPILERS

PDP-11 HOST	2000
PC DOS HOST	750
CP/M-86 HOST	750
CP/M-80 HOST	750
APPLE HOST	750

MANX SOFTWARE SYSTEMS
Box 55
Shrewsbury, NJ 07701
TELEX: 4995812

TO ORDER OR FOR INFORMATION:
CALL: 800-221-0440 (outside NJ)
201-780-4004 (NJ)

Australia: Blue Sky Industries — 2A Blakesley St. — Chatswood NSW 2067 — Australia 61-2419-5579
England: TAMSYS LTD — Pilgrim House — 2-6 William St. — Windsor, Berkshire SL4 1BA — England — Telephone Windsor 56747

Shipping: per compiler next day USA $20, 2 days USA $6, 2 days worldwide $75, Canada $10, airmail outside USA & Canada $20

UNIX is a trademark of Bell Labs. CP/M, CP/M-80 and CP/M-86 are trademarks of DRI. PC DOS is a trademark of IBM. MS DOS is a trademark of MICROSOFT.
N.J. residents add 6% sales tax.

CIRCLE 30 ON READER SERVICE CARD

COMPUTER LANGUAGE

CL PUBLICATIONS

ARTICLES

BASIC Becomes a Structured Language 20
by John G. Kemeny, Thomas E. Kurtz, and Brig Elliott
The founders of the BASIC computing language—John Kemeny and Tom Kurtz—discuss the past, present, and future of BASIC as a structured language and their new True BASIC compiler and interpreter.

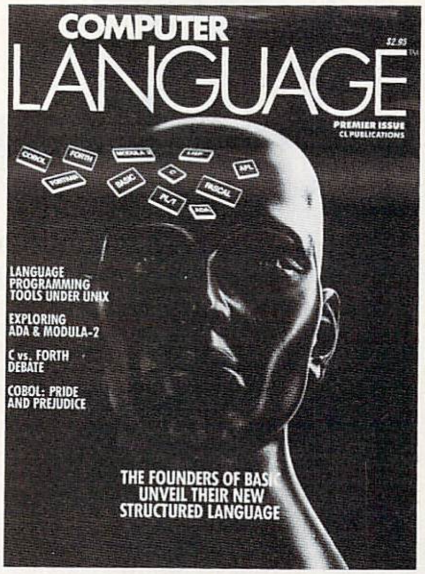
Programming in the UNIX Environment 27
by Bruce H. Hunter
Languages under UNIX have received much popularity in recent months. From C to RATFOR to "languages" like yacc, lex, Shell, and awk, the UNIX language family is unique because each language communicates easily with the other UNIX languages, and they all share a common function base.

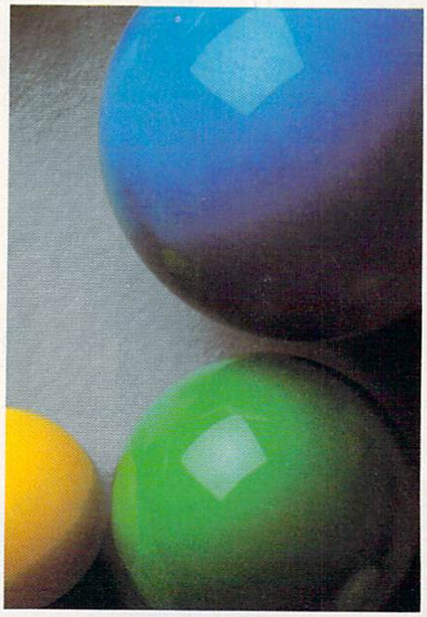
Bubble Sort, Insertion Sort, and Quicksort Compared 32
by Richard G. Larson
If you've been thinking that Bubble Sort is the only sorting algorithm available, consider what this author has to say about two more powerful approaches to sorting data.

Improve Your Programming with Structured Techniques 36
by Dr. Joseph B. Rothstein
From a disciplined and well thought out programming methodology comes a code design that tends to be more organized, reliable, and maintainable. This approach can help a programmer more clearly focus on the real problems and solutions involved in a given project.

COBOL: Pride and Prejudice 44
by Robert Wagner
According to this author, "revolutions do not succeed just because they are technically feasible. Failure of the old system is also required." He asserts that, while microcomputers and Pascal will probably be around for a long time, they will not replace mainframes and COBOL until they both become "senile."

Exploring Ada and Modula-2 51
by Namir Clement Shammas
Both Ada and Modula-2 have a number of new features and programming concepts in common, and their syntactic differences are also interesting to note. Here we take a critical look at both languages from five separate angles: data types, identifiers, program flow control, functions and procedures, and exceptions.

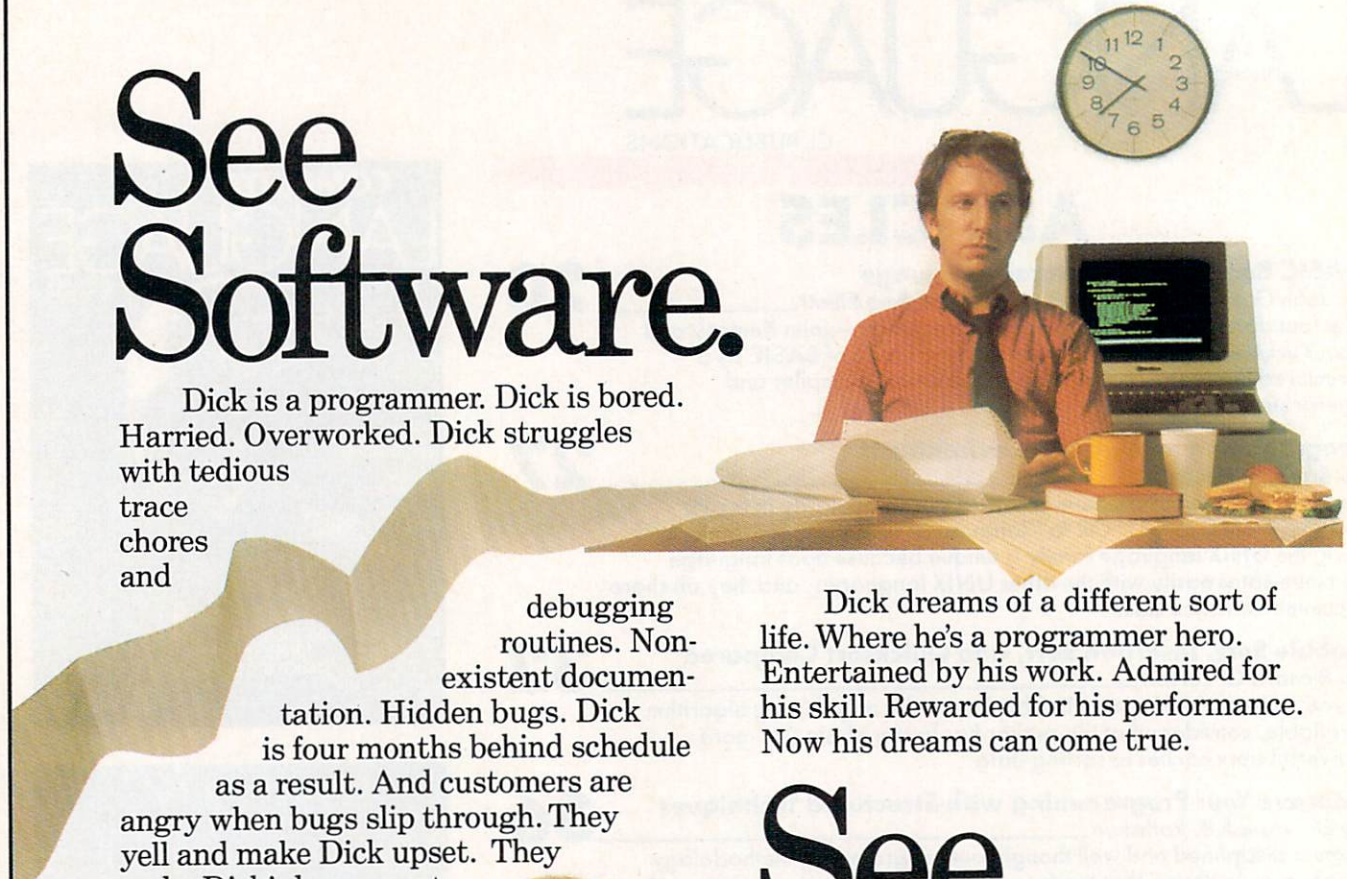

COMPUTER
LANGUAGE
$2.95
PREMIER ISSUE
CL PUBLICATIONS
COBOL
FORTH
MODULA-2
LISP
FORTRAN
BASIC
C
PASCAL
APL
PL/1
ADA
LANGUAGE PROGRAMMING TOOLS UNDER UNIX
EXPLORING ADA & MODULA-2
C vs. FORTH DEBATE
COBOL: PRIDE AND PREJUDICE
THE FOUNDERS OF BASIC UNVEIL THEIR NEW STRUCTURED LANGUAGE



DEPARTMENTS

Editor's Notes 7
Programmer's News Wire 9
Back to the Drawing Board 11
Designers Debate 13
Public Domain Software Review 61
Exotic Language of the Month Club 65
ComputerVisions 69
Software Reviews 71
Advertising Index 80

See Software.



Dick is a programmer. Dick is bored. Harried. Overworked. Dick struggles with tedious trace chores and debugging routines. Nonexistent documentation. Hidden bugs. Dick is four months behind schedule as a result. And customers are angry when bugs slip through. They yell and make Dick upset. They make Dick's boss upset. Nobody is very happy.

Dick dreams of a different sort of life. Where he's a programmer hero. Entertained by his work. Admired for his skill. Rewarded for his performance. Now his dreams can come true.

See Software Run.

Jane is a happy programmer. She uses ANIMATOR.™ It's a unique VISUAL PROGRAMMING™ aid for MICRO FOCUS™ COBOL.™ It runs on Jane's friendly microcomputer. It makes child's play of test and debugging tasks.

With ANIMATOR Jane sees a picture of the program explaining itself. In live action. In real time. In COBOL source code. As ANIMATOR displays the program listing, the cursor tracks the exact execution path. Including subroutine branches.

The view is precise. Compact. Unambiguous.

Jane can have the program run fast. Or slow. Or stop it. All at the touch of a key. This makes it easy to spot problems. Insert fixes. Set breakpoints. Examine details. Instantly. Because of ANIMATOR's sophisticated debugging commands.

ANIMATOR gives Jane more freedom to innovate too. Her programs are best sellers. They're delivered on time. With no hidden bugs. Jane's boss likes this about Jane. Because he doesn't like customers to yell at him.

Run, Software, Run!

This software vendor just went public. Because he doubled productivity. Eliminated bugs. Cut costs. Compressed development cycles. Produced terrific applications. Beat the competition to the market. And customers don't yell at him anymore. All thanks to ANIMATOR.

See ANIMATOR now.

Let ANIMATOR help speed your applications to market. ANIMATOR makes COBOL programs easy to comprehend. Fun to develop. A snap to maintain. Without the drudgery, you'll be happier. You'll do better work. In record time. This will make you richer. Faster. And your customers will stand up and cheer.

Write or call for a demo. Or detailed information. Right now.

MICRO FOCUS™

2465 East Bayshore Road, Suite 400
Palo Alto, CA 94303
(415) 856-4161

© 1984 Micro Focus, Ltd. All Rights Reserved.

LEVEL II COBOL, ANIMATOR, VISUAL PROGRAMMING, MICRO FOCUS and the MICRO FOCUS Logo are trademarks of Micro Focus Ltd.

To: Micro Focus
2465 E. Bayshore Rd., Suite 400
Palo Alto, CA 94303
I'd like more information

CL-5/84

Name__________________ Title__________________

Company__________________ Phone__________________

Address____________________________________

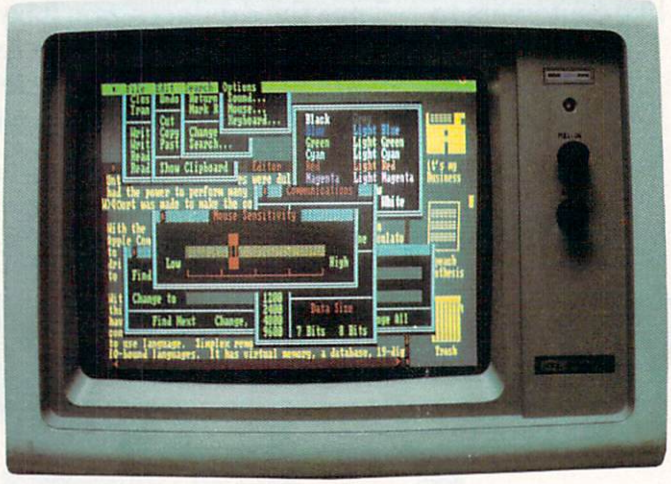
City__________________ State______ Zip__________

CIRCLE 32 ON READER SERVICE CARD

We thought about calling it MacSimplex . . . after all it makes your IBM®PC behave like a Macintosh™ and much more . . .

and with over two years in the making, the Simplex Database Management System has features like 32-megabyte virtual memory and the most powerful networked/relational database in the microcomputer industry. Simplex was designed around how you think and the Macintosh way, so that you can use your favorite mouse to handle those mundane tasks like menu selection and data manipulation. And, if you don't have a mouse, you can use our keyboard mouse simulator, MouSim™.

Pop-up and pull-down menus, dialog and alert boxes are not just added features, they are the heart of the Simplex way. In addition, Simplex gives you both a software and a hardware floating point capability, each with 19-digit accuracy. It permits login, password, privilege, and can be used on a local area network. Simplex has full communications and a remote or local printer spooler. Above all, Simplex is modular and grows with you! Simplex also has a full-featured, English-like language which is simple to use.



You can't buy Simplex™, but it is now available as an integral part of *it's my* ***Business***™ and will be used by *it's my* ***Word***™, *it's my* ***Graphics***™, . . .

Businessmen! *it's my* ***Business*** will revolutionize the way that you handle your business. It saves time, money, and standardizes your system for all who use it. *it's my* ***Business*** comes with applications like accounting, interoffice or intraoffice mail, editing, invoicing, inventory managment, mail list, calendar, scheduler, forms and more. You can modify each of these to create applications specifically designed for you... maybe we should have called it "it's your Business".

Professionals! *it's my* ***Business*** has over 200 pages of examples and demonstrations to show you how to solve your everyday professional problems. And if these examples aren't enough, we give you a complimentary one-year subscription to Questalk™, our hands-on Simplex applications magazine.

System integrators and consultants, beware! If you are not using *it's my* ***Business*** with Simplex to solve your problems, don't be surprised when more novice programmers solve that complex math, industrial engineering, or business problem faster. We think that you can cut your concept-to-development time by an order of magnitude!

it's my ***Business*** (includes *it's my* ***Editor***) - $695.00
it's my ***Business*** Demo Disk - $20.00
it's my ***Editor*** - $100.00.

Quest Research software is available through your local computer store or through mail order from Quest Software Corporation at (205) 539-8086. 303 Williams Avenue, Huntsville, AL 35801.

Value added resellers and dealers please contact Quest Research, Incorporated at (800) 558-8088. 303 Williams Avenue, Huntsville, AL, 35801.

Quest™

Quest Research Inc.

IBM is a registered trademark of International Business Machines. Macintosh is a trademark of Apple Corporation. *it's my* ***Business***, *it's my* ***Word***, *it's my* ***Graphics***, *it's my* ***Editor***, *it's my* ***Home***, *it's my* ***Voice***, *it's my* ***Ear***, *it's my* ***Statistics***, Simplex, MouSim, Questalk, and the Quest logo are trademarks of Quest Research, Incorporated.

CIRCLE 45 ON READER SERVICE CARD

Editor's Notes

As we present this premier issue of *COMPUTER LANGUAGE*, we look ahead with a positive and confident feeling that our new magazine will fill an important niche in the programming community.

We believe that for a long time there has been a real need for a magazine like *COMPUTER LANGUAGE*. In fact, during the past few months we haven't received just strong support for the magazine's concept; we've gotten a *phenomenal* amount of encouragement in virtually every area from editorial submissions to circulation response to advertiser sign-ups. Our main goal is to provide a well organized monthly magazine that focuses on the changing issues and trends in computer programming.

We hope you'll agree that our premier issue contains a balanced mix of interesting topics. In the magazine as a whole, we will always strive for *breadth*. In each article and department, we will encourage the author to convey the *depth* of a given subject in a clean, organized, and understandable way. And, to make certain that *COMPUTER LANGUAGE* becomes a servant of its readership, the publisher, Carl Landau, and I have decided to place a response time period of three months between this premier issue and the second issue. In October, *COMPUTER LANGUAGE* will go monthly, incorporating all the editorial feedback received from the premier issue.

One of the things I've enjoyed has been putting together the *COMPUTER LANGUAGE* bulletin board computer system. By using your computer (or terminal) and a modem, you can call into our remote CP/M-based computer at any hour of the day or night (phone: 415-957-9370) to:

- Write us an instant Letter to the Editor
- Download a program listing that was referred to in the magazine
- Submit an article you've written that you'd like us to consider for publication
- And a lot more . . .

About 70 percent of the material published in this premier issue was received electronically through the bulletin board. Operating with this kind of system allows us to have a production turn-around time of three weeks. *COMPUTER LANGUAGE*, as a result, will be much more current than most monthly magazines could ever become.

And it's all possible because, as magazine publishers, we are in the rare position of having readers who are able to telecommunicate their material.

Many people have asked us to devote a department to their own favorite language, operating system, hardware, etc. But since our primary goal is to become an objective technical forum, we've decided to set up our monthly departments so that each one serves as a mechanism for focusing in on important issues in programming, rather than allow a department to become a soapbox for a particular faction in the programming community.

In our *Back to the Drawing Board* department, readers can write in and present their technical questions, problems, complaints, etc. And in our *Designers Debate* department you'll find a lively discussion on controversial issues in programming. Our *Exotic Language of the Month Club* department provides the space for a different contributing editor each month to talk about a small language that utilizes interesting features worth considering.

The *Public Domain Software Review* is a very special department. Here, we'll look into the amazing world of public domain software and keep you aware of new programs worth downloading and others worth avoiding altogether. Also, in our *Programmer's News Wire* department you can keep up to date on the latest compilers, utilities, hardware, and conference proceedings of interest to technical people. In *ComputerVisions*, we'll sponsor an interview with a notable person in the technical community who will reveal his or her insights into the future of computing.

The publisher and I were both previously involved in a magazine many of you may have heard of—Dr. Dobb's Journal. We both appreciate the role this and other magazines have played in the evolution of the technical world of personal computing. But Carl and I are not trying to recreate any past traditions with *COMPUTER LANGUAGE*. We simply want to provide a useful magazine that has a fresh, clean style.

We started *COMPUTER LANGUAGE* without the financial backing of a large publishing venture, so we must depend on advertising and subscription income to make the magazine grow in the coming months and years. And, to do that, *COMPUTER LANGUAGE* has to be more than just good. It has to be great.

What you read in the pages ahead is the result of many people's collective, caring effort. In the back of all of our minds, I think we've all felt that the programming community is ready for *COMPUTER LANGUAGE*. We're anxious to hear what you think . . .

Craig LaGrow
Editor

COMPUTER
LANGUAGE

EDITOR
Craig LaGrow

MANAGING EDITOR
Regina Starr Ridley

TECHNICAL EDITOR
John Halamka

EDITORIAL ASSISTANT
Hugh Byrne

CONTRIBUTING EDITORS
Burton Bhavisyat, Leo Brodie, Ralph Griswold,
Ron Jeffries, Tim Parker, Ken Takara

CIRCULATION COORDINATOR
Renato Sunico

ART DIRECTOR
Jeanne Schacht

COVER DESIGN
Dow & Clement

PRODUCTION
Katherine Monahan

TECHNICAL CONSULTANT
Addison Simms

MARKETING CONSULTANT
Steve Rank

ACCOUNTING MANAGER
Lauren Kalkstein

PUBLISHER
Carl Landau
CL PUBLICATIONS, INC.

COMPUTER LANGUAGE *is published monthly by CL Publications, Inc., 131 Townsend St., San Francisco, CA 94107. (415) 957-9353.*

Advertising: For information on ad rates, deadlines, and placement, contact Carl Landau, Publisher, at (415) 957-9353, or write to: CL Publications, 131 Townsend St., San Francisco, CA 94107.

Editorial: Please address all letters and inquiries to: Craig LaGrow, Editor, COMPUTER LANGUAGE, *131 Townsend St., San Francisco, CA 94107.*

Subscriptions: Contact COMPUTER LANGUAGE, *Subscriptions Dept., 2443 Fillmore St., Suite 346, San Francisco, CA 94115. Single copy price: $2.95. Subscription prices: $24.00 per year (U.S.). Additional postage outside the U.S.: $12.00 per year surface, $30.00 per year airmail (U.S. currency only). Please allow six weeks for new subscription service to begin.*

Postal information: Second-class postage rate is pending at San Francisco, CA and additional mailing offices.

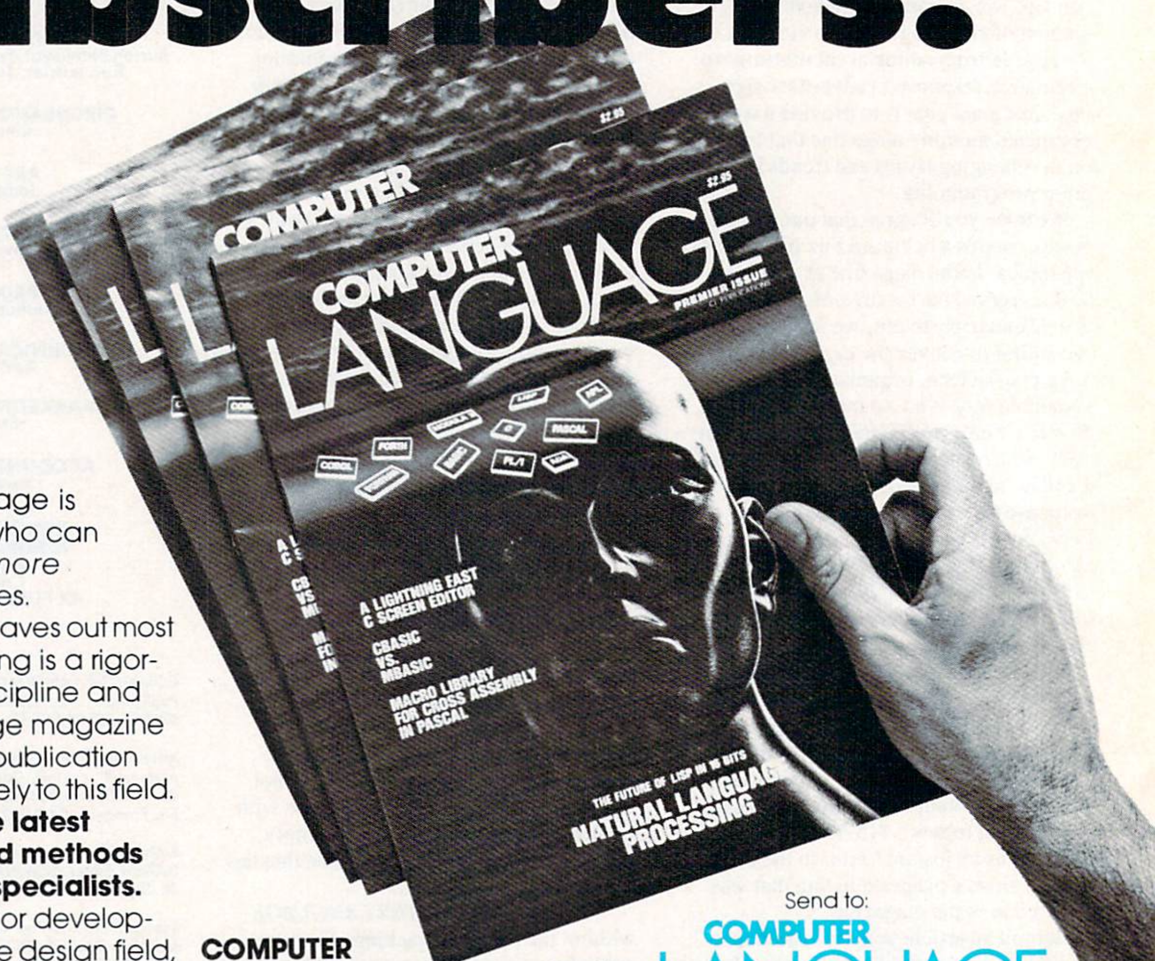
Reprints: Copyright 1984 by CL Publications, Inc. All rights reserved. Reproduction of material appearing in COMPUTER LANGUAGE *is forbidden without written permission.*

Change of address: Please allow six weeks for change of address to take effect. POSTMASTER: Send change of address (Form 3579) to COMPUTER LANGUAGE, *131 Townsend St., San Francisco, CA 94107.*

COMPUTER LANGUAGE *magazine is typeset by Mercury Typography, Inc., San Francisco, CA, and printed by Brown Printing Company, Inc., Waseca, MN.*

COMPUTER LANGUAGE *is a registered trademark owned by the magazine's parent company, CL Publications. All material published in* COMPUTER LANGUAGE *is copyrighted © 1984 by CL Publications, Inc.*

We're looking for a few good subscribers.


COMPUTER
LANGUAGE
$2.95
PREMIER ISSUE
COBOL
FORTH
MODULA-2
C
PASCAL
APL
BASIC
PL/I
A LIGHTNING FAST C SCREEN EDITOR
CBASIC VS. MBASIC
MACRO LIBRARY FOR CROSS ASSEMBLY IN PASCAL
THE FUTURE OF LISP IN 16 BITS
NATURAL LANGUAGE PROCESSING

Computer Language is written for people who can program in two or *more* computer languages.

Let's face it, that leaves out most people. Programming is a rigorous, intellectual discipline and Computer Language magazine is the first and only publication dedicated exclusively to this field.

Your source for the latest technical skills and methods used by software specialists.

We cover the major developments in the software design field, from theory to implementation. Computer Language focuses on the most important and useful language design information available in the fast-moving microcomputer industry.

Written for the person who takes computing seriously.

We're talking about you — the experienced software author, programmer, or engineer who routinely programs in two or more high-level languages. A person who understands the creative nature of programming and appreciates the beauty of efficient code in action.

COMPUTER LANGUAGE will constantly challenge your abilities.

The foremost industry experts will discuss: · Algorithmic Approaches to Problem Solving · Language Portability Features · Compiler Designs · Utilities · Artificial Intelligence · Editors · New Language Syntax · Telecommunications · Language Selection Criteria · Marketing Your Own Software · Critical Software & Hardware Reviews

Plus, columnists and reader forums that will put you in touch with the latest developments in the field.

Send to:
COMPUTER
LANGUAGE
2443 Fillmore Street Suite #346
San Francisco, CA 94115

YES! Start my charter subscription to Computer Language. My 1 year charter subscription is just $19.95, a $15 savings under the single copy price. Guarantee: I can cancel my subscription at any time for a full refund.

☐ $19.95 Payment Enclosed ☐ Bill me.

Name

Address

City State Zip P

PROGRAMMER'S NEWS WIRE

By Ron Jeffries

My goal with this column is to serve as your "designated information junkie." That means that each month I'll be collecting tidbits of news about the computer industry as it relates to the technical interests of software authors and systems programmers.

As you are probably already painfully aware, an immense amount of info gets published every month about computers and the computer industry. This became clear to me recently when a large consulting job kept me away from my reading for about three weeks. When I returned to some semblance of sanity (and my office), I faced a stack of magazines, trade papers, newsletters, and press releases that was over five feet tall. None of us can really keep up with the constantly increasing flow of raw data, but it's a challenge at least to try.

I look forward to writing this column because the readers of *COMPUTER LANGUAGE* are likely to be the kind of knowledgeable, technical people who enjoy keeping up with the important issues and trends in programming and hardware design. Many of you are dedicated, hard-core personal computer enthusiasts. This column will include a mixture of news items and pointers to articles, upcoming product announcements, conference proceedings, and other sources of information related to programming and computer languages that may be somewhat off the beaten path—or, should I say "off the beaten TRAC?"

If you don't find mention of your favorite subjects here, take a minute and drop me a note or, better yet, use the *COMPUTER LANGUAGE* remote CP/M bulletin board computer (phone number: (415) 957-9370) to leave me a message. For example, has anyone out there ever used Ralph Griswold's ICON (a worthy successor to SNOBOL) on the IBM PC? Maybe we'll have some answers in A Future Column. I hope you'll use the *COMPUTER LANGUAGE* bulletin board to let me know of interesting news, technical articles in obscure journals, etc. I can also be contacted via The Source (ID CPA025), or you can telephone me at (805) 967-7167.

Digital Equipment Corp. (DEC) is working on something called the MicroVAX II, a four-chip version of the popular VAX 11/780 superminicomputer. Altogether, the MicroVAX II chip set contains 1,220,550 transistors. DEC has not officially announced a projected delivery date for the new VAX, but mid-1985 looks likely. The previous MicroVAX I has been in production since March 1984, and is coming off the production line at a rate of about 250 per month. The MicroVAX II will have hardware floating point and should provide about twice the performance capability of the MicroVAX I. A stripped down version of the standard VMS operating system (known as MicroVMS) will be offered for both machines. Although originally due to be released with the first production MicroVAXs, MicroVMS has been delayed until the fall of 1984.

Speaking of delays, **Ovation Technologies Inc.** has once again delayed the introduction of its integrated software package called Ovation. The company's revised schedule calls for Ovation to be shipped by mid-October. Two other integrated software packages are being launched this summer—Symphony by Lotus Development and Framework by Ashton-Tate Inc.

As an indicator of how competitive the personal computer software market has become, consider the case of **Leading Edge Products Inc.** When IBM announced the DisplayWrite I and DisplayWrite II packages, Leading Edge cut the price of its word processing package from $295 to $200. Two weeks later, IBM announced another series of inexpensive software for the PC and the PCjr, including yet another inexpensive word processor. The next day, Leading Edge dropped the price of its word processing program again—this time to a retail price of only $100.

Expect to see a revised version of the **Apple Macintosh** early in 1985, with expanded memory and probably 256K to 512K of RAM. Already selling at a rapid clip, the enhanced Mac should prove to be a serious competitor to the IBM PC.

The so-called "Fat Mac" isn't IBM's only worry. **Hewlett-Packard** gives every indication of being deadly serious about selling a significant number of IBM-compatible computers. Although the touch-screen HP-150 hasn't set the world on fire, the new notebook-size model called "The Portable" may do much better. For just under $3,000, the new HP is a lightweight (under 10 pounds) portable with a 16-line by 80-column liquid crystal display. It has the MS-DOS operating system in ROM as well as the Lotus 1-2-3 spreadsheet, a simple word processor, and terminal communications. Similar machines are expected over the next several months from IBM, Kaypro, and several other vendors.

COMAL (COMmon Algorithmic Language) is a little known programming dialect of BASIC that is popular in Europe. Best described as a hybrid of BASIC and Pascal, COMAL might also be called "BASIC without tears" (or at least without line numbers). COMAL was created by Borge Christensen of Denmark in 1971 when his school purchased a Data General Nova 1200. Commodore released a public domain COMAL for the popular C64, but a more advanced (and proprietary) version 0.14 is estimated to have over 50,000 users. Although mainly a Commodore phenomenon so far, COMAL for the Apple and the IBM PC is also under development. An active COMAL User's Group USA Ltd., headed by Len Lindsay, publishes *COMAL Today*. Issue no. 2 had 36 pages of information about the language and a number of sample COMAL programs. A subscription to the newsletter is $14.95 in the U.S. from *COMAL Today*, 5501 Groveland Terrace, Madison, Wis. 53716.

Q'Nial (Queen's University Nested Interactive Array Language) is a new language designed by M.A. Jenkins and Trenchard More. Using a model of nested array data objects, the Nial language borrows ideas from APL, Lisp, and structured programming. The Q'Nial interpreter is written in C and was originally developed on the UNIX operating system. It has been ported to a half-dozen other systems, including the IBM PC and the IBM 4341 mainframe. While running a Q'Nial program you can call UNIX utilities to create or modify files that can then be processed by Q'Nial. For more information, contact Nial Systems Ltd,, Box 2128, Kingston, Ont., Canada K7L 5J8, (613) 549-1432.

When the **IBM XT/370** was announced last fall, few people realized that most of the technical background on the inexpensive implementation of the 370 architecture had been published over a year earlier. In the July 1982 issue of the *IBM Journal of Research & Development* (Vol. 26, No. 4, pp. 401-412) an article written by P.W. Agnew and A.S. Kellerman titled "Microprocessor implementation of mainframe processors by means of architecture partitioning" appeared. The authors outlined seven approaches for implementing a complex architecture using large-scale integrated circuit technology. Even though the XT/370 cat is out of the bag, the article still makes for good reading since it covers other alternative solutions in enough detail to let the reader speculate about other low-end System/370 implementations IBM may be developing. If nothing else, the concluding sentence is intriguing, ". . . the 10-year trend of implementing successively higher-level architectures within a single microprocessor chip should be continued to and then beyond the architecture levels implemented by today's mainframes."

General Electric has developed a contrast enhancement technique using a photo-bleachable dye coating that is applied to semiconductor wafers at the beginning of the fabrication cycle. By using the coating, GE researchers have made experimental chips with line widths of 0.4 micron, using commercial optical projection equipment known as a "stepper aligner." This is about half the line width that can be produced with this equipment without the coating.

Digital Research and **Softech Microsystems** have decided to adapt their operating systems to the popular PC-DOS from IBM/Microsoft. By mid-summer, Softech plans to allow p-System applications to execute within a PC-DOS or MS-DOS environment. PC-DOS will act as a "host," with the p-System running as an application (only this particular application will just happen to be an operating system). This will allow software developed originally for the p-System to run under the much more popular PC-DOS. Digital Research has taken a different approach with what they call the "Concurrent PC-DOS" operating system. This reincarnation of Concurrent CP/M can run up to four PC-DOS or CP/M programs at one time. It requires a minimum of 256K of memory, although 512K and a hard disk are recommended for maximum efficiency. This multitasking capability is not yet available with the standard Microsoft DOS. The new operating system will be released in the third quarter and will cost $295.

US Software offers single and double precision floating point software using the IEEE K-C-S draft standard. The package is available in source code for six processors (8086, 8051, 8085, Z80, 6809, and 68000). The same folks also make a small, efficient multitasking kernel which is also sold with full source code. For more information, contact them at 5470 N.W. Innisbrook Place, Portland, Ore. 97229, (503) 645-5043.

Two good new magazines are out that you should know about. ***UNIX/World*** is a glossy, nice-looking journal devoted to the UNIX operating system. They get points in their first issue for running a long satirical piece by Brian Boyle which pokes fun at UNIX. They have an impressive stable of writers who are knowledgeable about UNIX. Subscriptions are $18 a year from Tech Valley Publishing, 289 South San Antonio Road, Los Altos, Calif. 94022. Another worthwhile new publication is ***IEEE Software.*** It has the editorial look and feel of its respected sister magazine *Computer*. Definitely aimed at software professionals, this quality magazine is full of outstanding articles and is carefully edited. It is published quarterly by the IEEE Computer Society, 10662 Los Vaqueros Circle, Los Alamitos, Calif. 90720 and costs $10 to IEEE members.

Trilogy Ltd. has abandoned its goal of producing a powerful computer compatible with IBM mainframes. Trilogy's strategy was to integrate a large number of chips on a single 2.5-in. diameter silicon wafer. For example, a complete IBM-compatible mainframe CPU was supposed to be fabricated on only nine wafers! However, because of technical problems with the wafer-scale integration, the delivery date for the completed Trilogy system was missed several times. Ultimately, the delays pushed the projected availability of the new system so far into the future that it seemed unlikely that the Trilogy computer would be competitive with new products expected from IBM. The company now plans to continue its development of wafer integration, and may decide to build a less-ambitious computer using the wafer components. Gene Amdahl founded Trilogy in 1980 after leaving Amdahl Corp.—another IBM plug-compatible company he started after leaving IBM, where he was the chief architect for the IBM System 360.

Jack Tramiel, former head of Commodore, is starting a new firm called **Tramiel Technology Ltd.**, or "TTL" for short. He has attracted several former Commodore employees as well as his two sons and his brother to the new company. Rumor has it that they may be planning a personal computer based on the National 16032 chip.

The **Sinclair** QL $500 68008-based machine may be available in the U.S. by late 1984. This all-in-one computer has 128KB of RAM, two tiny tape drives which store 100KB each (worst-case access is seven seconds), and a single-user, multitasking operating system. BASIC programs are stored as ASCII files (i.e. they are not tokenized). This means a program can create another program on tape and then execute it. However, at the Summer CES, the BASIC had very slow benchmark timings, taking 19.5 seconds to run a FOR loop from 1 to 10,000.

Well that's it for now. Drop me a line when you can!

BACK TO THE DRAWING BOARD

By Burton Bhavisyat

Problem solving by reader feedback is what this column is all about. Back to the Drawing Board is your link to the world of *COMPUTER LANGUAGE* and its readers. This column is meant for you. It's your way to communicate to a vast group of computer programming people and to help others by solving their difficulties.

We want to help overcome the perplexing problems caused by the lack of knowledge. But we're not about to try providing such information the same way other magazines do. They set up one person as the "expert" who tries to find answers about anything and everything asked. We consider that to be a practically impossible task.

This is a reader participation column. Each month we will discuss a series of good questions that have been posed by our readers. And for the previous month's problems, we will provide the solutions that are sent in by our readers—i.e., you.

When you see a problem that you think you can resolve, please share your knowledge. If you have questions that are begging to be answered, please feel free to ask.

There is no limit to the definition of "problem" either. Problems may range from hassles with a specific piece of software all the way up to gripes about the computer industry in general. And if you feel the need to air an opinion that could be considered controversial, don't be afraid to express yourself. We are only biased in favor of one group—the readers of *COMPUTER LANGUAGE*.

A large percentage of our readers are well educated (many are Ph.D.'s), fluent in computer languages, quite experienced (anywhere from 2 to 30 years), and vastly opinionated—not to mention friendly, knowledgeable, and willing to help.

Each month in Back to the Drawing Board we'll consider issues raised by such readers. This issue's column is your invitation to participate in a lively and informative dialogue.

Since this is the premier issue, we don't have much feedback—yet—but we do have the results from a survey of representative members of the professional technical community. You will be happy to know that the world is not made up entirely of anxiety-ridden, neophyte computer hobbyists sitting intimidated in front of their screens as the majority of glossy magazines on the newsstands would have you believe. An enormous number of competent and qualified computer people exist—they are the target of this magazine.

We asked some of them their views on the idea behind *COMPUTER LANGUAGE*, and we got some very interesting replies.

Richard Larson of Oak Park, Ill., wrote, "It's a bit horrifying to see yet another computer magazine appearing—worse yet, a possibly interesting one."

Linda Wilson of Billerica, Mass., expressed the opinion of most of the respondents, "Your magazine has the potential to fill a substantial information gap among existing computer publications."

Similarly, Juan A. Navarro of McLean, Va., said, "This magazine is a much needed service for the professional computer person who wishes to avoid the hype associated with currently available publications (except for a few). I am willing to be involved."

When analyzing our survey responses, we were pleased to discover that what the readers of *COMPUTER LANGUAGE* are hoping to see is just what we want to provide. If you know something about the topics on the following list, which was compiled from survey suggestions, please consider sharing your knowledge with our readers. Each item is credited to one source (as will be our practice), but the desire to see each of the topics covered in the pages of *COMPUTER LANGUAGE* was widespread. We hope that the list excites you as much as it does us.

Loren Amelang, of Philo, Calif., feels "hardware handshaking is a much neglected subject" and advises, "Don't get too caught up in the 16-bit hype; the Z80 still does more work of more kinds and often does it faster."

Not everyone agrees with that one. Do-While Jones of Ridgecrest, Calif., says, "Now everybody seems to be interested in the IBM PC, and nobody talks about CP/M-80 anymore . . . I doubt that I will write any more CP/M articles."

Anton Dovydaitis, Santa Cruz, Calif., would like to see discussions on 68000 assembly, programming icons, and windows, plus good searching and sorting techniques.

Ronald Gutman, San Jose, Calif., is hoping for an article about choosing symbolic names to improve programming productivity.

Marlin Meier, Mayfield Heights, Ohio, is intersted in a C library with specific functions dedicated to compiler writing. Now there's an idea that could be expanded upon any number of ways: standard libraries designed for standard generic applications would certainly make a programmer's life easier.

Richard Rodman, Falls Church, Va., expressed his desire for articles on CP/M BIOS modification and construction. This would fit in nicely under the general topic of operating systems: design and customization.

A number of good topics for discussion were brought up by George F. Reeves, Decatur, Ga., who wrote that his interests included "random access techniques for files, algorithms for computing, sorting techniques, development of macro libraries and their use, communications protocols, customization of CP/M's BIOS, multiprocessing techniques, software protection and encryption, and data compression."

Database management, B-trees, and file handlers are areas of concern for Chris Deppe, Woodland Hills, Calif.

On the subjects of artificial intelligence and expert systems, which we are anxiously waiting to read about, John E. Staneff Jr., Ellensburg, Wash., states, "Even the most complicated algorithms based on post-doctoral mathematics

C

Software Development PCDOS/MSDOS

Complete C Compiler
- Full C per K&R
- Inline 8087 or Assembler Floating Point, Auto Select of 8087
- Full 1Mb Addressing for Code or Data
- Transcendental Functions
- ROMable Code
- Register Variables
- Supports Inline Assembler Code

MSDOS 1.1/2.0 Library Support
- All functions from K&R
- All DOS 2.0 Functions
- Auto Select of 1.1 or 2.0
- Program Chaining Using Exec
- Environment Available to Main

c-window™ Symbolic Debugger
- Source Code Display
- Variable Display & Alteration Using C Expressions
- Automatic Commands
- Multiple Breakpoints by Function & Line Number

8088/8086 Assembler
- FAST — Up to 4 times Faster than IBM Assembler
- Standard Intel Mnemonics
- Compatible with MSDOS Linker
- Supports Full Memory Model

8088 Software Development Package $199^00

Includes: C Compiler/Library, c-window, and Assembler, plus Source Code for c-systems Print Utility

c-systems

P.O. Box 3253
Fullerton, CA 92634
714-637-5362

CIRCLE 14 ON READER SERVICE CARD

should be able to be expressed in common English."

"Of greatest interest are communications programs and/or software and hardware that will allow dissimilar equipment to exchange data files," notes Charles Ballinger, Spokane, Wash.

A question asked (and answered) by Paul Gans, New York, N.Y., is, "Do we need a high-level graphics language that interfaces to suitable graphics drivers? You bet we do, and DRI's GSX is not the answer!"

John Synde, Finksburg, Md., is "interested in application requirements and programming techniques which influence language, e.g., recursion, structured code, matrix manipulation, artificial intelligence, graphics, etc."

That was a list of some specific ideas that readers hope will be treated by *COMPUTER LANGUAGE*. Equally evident in our survey was that programming professionals tend to be quite opinionated. Here are some viewpoints that have been expressed about the future content of *COMPUTER LANGUAGE:*

Lawrence Hughs, Tallahassee, Fla., writes, "Try not to get hopelessly lost in trendy issues like Ada and Forth. Stick to hard-core stuff."

Morton Kaplon, Pomona, N.Y., says, "It might be interesting to survey readers to elicit what they think would be useful benchmarks for a variety of programs."

Professor David Ross of the University of Iowa says he would really be interested in seeing an article, collection of articles, or monthly column on programming aesthetics. "Such an article would include discussions and examples of programs whose elegance and all-around rightness are entirely self-evident," notes Ross. "Most experienced programmers have, I believe, a sense of this sort of aesthetics."

He also suggests "the magazine should keep an eye on others which intersect it in content and either cross-reference or repeat in condensed form the material contained therein." Ross adds, "I hope the magazine flies, especially now that *Byte* has abandoned programmers in favor of salesclerks."

A significant number of writers expressed the ideas of Richard Rodman, Falls Church, Va., who says, "People who write software for micros should not be so short-sighted as to think that the IBM PC and suchlike machines are anything more than slighty faster turtles. Computer manufacturers are now so 'consumesmerized' that they are turning out sizzle without contents, e.g., Macintosh—cutesy and useless. Someday the infantile computer industry will realize the need for a machine-wide interchange standard for media, from 370's to Cray-1's to SEL's to PDP-11's . . . but as long as they are applauded for incompatibility, it'll never happen—we'll be drowned in a sea of unsupported and incompatible glitter boxes."

If that one warmed your blood a little, let's consider an essay written by someone who sounds very familiar. One of the most fascinating replies to our survey was submitted by Dan Daetwyler, Springdale, Ariz.

"I'm a graduate physicist that drifted, early on, into data processing. I have well over 20 years of experience . . . and have used about 30 different languages. From your comments, I'm not sure if you intend to publish programs as well as articles. If you do plan to publish source code and can beat the size problem, I've several I would release for publication. Frankly, I'm a bit skeptical . . . as soon as a program becomes large enough to be interesting, it becomes too large for publication . . . I'm an assembler 'bigot' and find the debates (never ending) about which compiler is better than best a bit boring. Some year we'll learn that language is almost totally an emotional issue. Some minds work well in one language and not in another. To admit that the reason I don't like a particular compiler is because I don't think that way is to admit that my mind is limited! Heavens!"

Daetwyler goes on to state, "I spell cat with a 'K' (thank God for Word Proof), enjoy writing programs, and don't particularly enjoy writing documentation but always manage to get it done. I'm half of an electronics engineer, and I'm not the world's greatest/fastest programmer. I've known at least two guys who were better. But not three. I have, as you will have clearly noted, a particular turn of phrase that does not lend itself well to formal writing. I do usually manage to communicate. Usually by sheer verbosity . . . Yeah, I'm retired . . . the world just doesn't believe it yet."

The final note of the month comes from Roland Beander, San Rafael, Calif., who expressed the sentiments most computer professionals have toward *COMPUTER LANGUAGE*, "I am hoping that your magazine can become a forum for all of the issues to get kicked about and argued, and a place for some sanity to emerge!"

—Readers??

Forth Vs. C

By Ken Takara

Programming is a curious activity. On the one hand, it is a discipline of logic, both formal and practical. On the other hand, it is wracked by controversy over such topics as language preference, operating system environment, microprocessors, and techniques.

Perhaps it is a peculiarity of human nature that every possible topic has its adamant supporters and equally determined detractors. In any event, it is to this human trait that this column is devoted.

Each month, we will explore some realm of contention in the software world, considering such areas as ideal operating system environments, "universal" languages, or micros vs. mainframes. The field is broad, and everything is fair game.

If you have a topic that you want to bring into the arena, send us a letter or give us a call. You might have the opportunity to take the stand. And let us know if you think you can add to a topic that was presented here, for it is dissension that keeps this field alive and healthy.

This month, we present a discussion on the relative strengths and weaknesses of the two languages—C and Forth.

Both languages are in use in the mini- and microcomputer industry for systems development. Each has its own following of devoted (and sometimes fierce) proponents who proclaim the advantages of their chosen language with such fervor that one who is not initiated into the inner circle might wonder about the sanity of these programmers.

For this discussion, we are indebted to Edward Elhauge, Forth proponent, and to Russ Innes, C advocate. Both are software engineers, have experience with the languages, and are reasonably sane.

Forth and C are systems development languages—that is, they are generally used at a level very close to the machine for the purpose of building system utilities. In contrast, many other high-level languages such as Pascal, COBOL, or FORTRAN are used to build machine-independent applications. For this reason, a programmer may be expected to have a considerable knowledge of the machine or of the system architecture in order to use the language most effectively. This again is in opposition to the other high-level languages, which try to hide the machine from the programmer in order to achieve machine independence.

These languages are characterized by weak or nonexistent data typing, as opposed to the strong data typing provided by a language like Pascal. The data types available are relatively simple and unsophisticated and interface fairly easily with assembly language routines. Error checking during execution may be minimized in order to improve execution speed.

Weak data typing allows the programmer to view a variable in nearly any way he or she desires. For example, the same variable might be used as an integer in one part of the program and as a pointer (address) elsewhere. At the systems level, the programmer is often interested in the actual nature of the data being manipulated. Thus, sophisticated data types that hide the exact implementation of some information or data structure are absent. The exact implementation of such things as *BOOLEAN* types (provided by other high-level languages) are left up to the programmer in Forth or C.

Generally, the kinds of programs written in these languages include system utilities, support routines, or real-time and control programs. They reduce the need to program exclusively in assembler by providing a combination of the high-level language's expressiveness and the assembler's explicitness.

C was developed at Bell Labs by Dennis Ritchie for use on the DEC PDP-11 minicomputer. Many of the operations available, in fact, directly correspond to PDP-11 machine instructions. For example, the operator "++" is equivalent to the machine instructions, "fetch then increment the variable", or, "increment the variable then fetch". You might consider C to be a medium- to high-level implementation of PDP-11 assembler. Its syntactic similarity to Pascal makes it very easy to read as well as to learn.

Forth was developed by Charles Moore at the National Radio Astronomy Observatory as a language for developing telescopic device-control programs. The language allows you to write programs by creating "words" that each perform a certain basic function. For example, you might create a word called *TURN-MOTOR,* which turns the telescope's drive motor. This, along with other words, would be used to define a high-level word such as *POINT-TELESCOPE,* which performs each of the functions necessary to position the telescope to point to a particular celestial coordinate.

Q What kind of application would you use C for? And what might be considered a difficult task to implement in C?

A Innes: People use C for things like system utilities, where you need to perform primitive operations, such as manipulations on bits, registers, or machine-dependent memory locations. C's set of operators are similar to the PDP-11 machine instruction set. It's like a high-level assembly language, so you can use it in place of an assembler. The UNIX operating system developed at Bell Labs was written in C.

Combined with the appropriate operating system, you could use C for real-time applications. Unlike Forth, C is a compiled language. It runs much faster and can respond very quickly to a constantly changing environment. Most real-time operating systems take care of multitasking and will coordinate communications for you, and you can use these directly from C. Forth is less portable since it provides its own system, so you can't take advantage of existing systems that offer more efficient utilities for doing these things.

You probably wouldn't want to write business applications where complex data structures are used, especially when there are languages like COBOL or RPG that handle those problems much better. C lacks sophisticated string handling capabilities and BCD (binary coded decimal), and its file access methods are relatively primitive. All of these things are frequently used for business work.

Q How about Forth? Where is it best used, and what would be considered difficult to do in this language?

A Elhauge: Forth is good for writing control programs and real-time applications, which is the sort of thing it was designed for. With Forth you can easily manipulate small pieces of the program directly from the keyboard, without having to write special drivers for them. For instance, you could test the word that moves one motor on a robot. Or, if you wanted to monitor the effect of an interrupt handler or a specific task in a multitasking environment, you could easily do so without having to link any special modules. You can track problems while the program is running without halting it.

One of the hot things in business programs today is something called the "integrated environment." Forth is such an environment. You could build a set of business applications in Forth, and they will be able to communicate with each other. You simply compile the applications you want.

Of course, most business people find Reverse Polish notation a bit awkward, and the lack of string variables and real numbers make it far from ideal. You could do it, but it wouldn't be easy.

Q Can you describe some of C's strengths?

A Innes: C is both modular and structured. You can save object modules in a library and link them when needed. You can also make forward references—unlike Forth or Pascal, which require you to define everything before using it. Unfortunately, C doesn't let you nest subroutine definitions as Pascal does. They're all global, and that can be irritating when you want to restrict functions.

Unlike Forth, C is a compiled language. Where Forth's interpreter has to search down a linked list, C generates direct machine instructions, which make things go much faster in general. Since it is so close to assembler, you can use a number of operators that take advantage of machine-level functions to improve speed. In C, for example, the statement *x=y++* means

```
x:=y;
y:=y+1;
```

When the operator ++ follows the variable, it returns the value of the variable prior to incrementing. Alternatively, when it precedes the variable, it is incremented before returning the value. This reflects operations available at the machine level. The operator += is another example of a machine-level instruction in C. It generates a machine instruction of the form *INCR address,value.*

You can also specify register variables—that is, you can tell C that a variable should refer to a register, which is useful for loop counters. I suppose they did this to avoid having to write an optimizing compiler, which structures the generated code for you in the most efficient manner.

C has local variables, which Forth does not. The set of variables associated with the current subroutine call is dynamically allocated on a stack, so you can handle recursive functions easily. Also, unlike Forth, C works with subroutine parameters by name rather than by manipulating a stack. This makes C code much more readable since you can quickly see the underlying algorithm.

Q What are some of Forth's strengths?

A Elhauge: Forth is an extensible language. In most other languages, you would write subroutines that you call and extend in that way. But you can't actually make these new things an integral part of the language. In Forth, if you define a new word, you have added a new keyword to its vocabulary. (Listing 1 is an example of a few words from a Forth glossary.) This is how you program in Forth—by extending the language as you go. That's why you don't have all the functions that other languages provide for you. You build the ones you need.

Since Forth is based on a stack, the most natural way to handle expressions is in Reverse Polish (also called postfix) notation. With the standard prefix or inflix notation, the compiler or interpreter has to check ahead and evaluate the arguments used by an operator and then verify that the arguments are correct for that operator.

In Reverse Polish notation, if an argument is an expression, that expression will already be evaluated by the time the operator that needs them is reached. In fact, compilers often convert the infix or prefix expression to Reverse Polish notation while parsing.

You write Forth programs by defining new words. These may be functions, variables, or constants. Since Forth builds all its words in a uniform manner, once you've examined the structures of all these words, you see that they all function in the same way. Because of this, you can actually create new data types such as n-dimensional arrays or string variables and add them to the language.

Q Could you mention programming styles in C?

A Innes: C is a nice language to program in. For one thing, expressions are written in normal algebraic form. C has a limited macro-like facility—really a text replacement function. You can also include source files from a library. Using this feature, you don't have to rewrite frequently used routines.

C code looks a lot like Pascal, so if you know that language you'll find that C is very readable (printed here as Listing 2 and also available on the *COMPUTER LANGUAGE* bulletin board computer: (415) 957-9370). Of course, all the programming tricks that C gives you might encourage you to write some very obscure code, but you can abuse any language.

C is closely related to UNIX since it was originally developed to create UNIX. The UNIX operating system provides task communication using "pipes." A task can send data or messages to another task via these pipes. This protects tasks from inadvertently destroying each other while writing in another task's space.

A good part of C programming involves calls to the system to perform various tasks. Also, the system includes numerous libraries of small routines, programs, and other packages that can be called. There are many good packages available, including system utilities for doing disk-file management and full-scale relational data bases. So you don't necessarily need to write everything in C when you need it. The idea is to call upon the resources of existing software to help you out rather than develop it all each time you need it.

Q How about Forth programming styles?

A Elhauge: In Forth, the idea is to factor a routine into its functional subroutines, then factor each of these until you reach a

```
ROT  (a b c -- b c a)

     Remove the third item on the stack and put it on top.

"rote"

RP!      ( -- )

     Clear the return stack by re-setting it to the
address held in the variable, RO . "r-p-store"

                         (Description)

    A  "glossary"  is  the  Forth  programmer's   language
reference  and  consists  of a list of  available  words  in
alphabetical  order,  and gives the word,  a stack  picture,
explanation,  and,  in some cases, pronunciation.  The stack
picture shows the state of the stack on entry to the word to
the left of the "—" and its state on exit to the left.  The
stack is read from left to right,  with the lowest  relevant
item  to  the left.   Note that Forth words may contain  any
characters  except the "blank," which is used as  the  token
delimiter.   You  could even define numbers as Forth  words.
In fact, certain values such as "0" are defined as constants
in the Forth dictionary.
```

Listing 1.

```
/* Include files for time routines */
#include <stdio>
#include <ssdef>

*           TIME_NONE - No timeout on clock
*/

time_wait(   )    /* Function returns integer, no argruments
passed */
   {
   int status;         /* To return status of call    */

   if ( TimeStatus == TIME_RUNNING )
     {
     sys$waitfr( TIME_EF );
     status = TIME_OK;
     }
   else
     {
     status = TIME_NONE;
     }
   return ( status );
   }
```

Listing 2.

continued

primitive level. Then you code starting with the primitives and work your way up. One aspect of Forth is that it's quite terse. You like to keep the size of each routine small. You also need to use a lot of comments because of its brevity.

Forth doesn't use disk files. The disk is organized into "screens," where each screen is 1,024 bytes long and arranged into 16 lines of 64 characters each. You are required to fit definitions into these blocks so that no definition is more than one block long. But you often find yourself trying to cram your code to fit the screen limitations, and this can impair readability (printed here as Listing 3 and also available on the *COMPUTER LANGUAGE* bulletin board service).

Q What kinds of problems do you find with C?

A Innes: Well, the terse symbols that make coding fast can encourage some people to write unreadable code. This is true if you use some of the tricks provided. For example, an expression like $a=b++/n++$ is really cryptic. C's string functions are limited; it lacks some of the sophisticated data structures that other languages offer and its file access methods are primitive. Also, if you have all the libraries you would want, it requires a large chunk of disk space. However, it is well-supported and easily portable, which are important in software development.

Q What sort of problems do you see with Forth?

A Elhauge: Forth has a serious lack of standard utilities. The attitude is, "If you need it, build it." This leads to a certain amount of duplicated effort. Some utilities are available, but they are distributed as source, and may require modification to make it run on your system because of slight dif-

(Description)

This example shows a function written in C that returns integer values. In C, all arguments are passed by value. In order to pass by reference, you have to actually pass the address as a value. The symbols "{" and "}" (braces) are used to delimit a begin...end block. Comments are delimited using "/*" and "*/" in a manner similar to PL/I. Note that "=" is used for assignment while "==" is the comparison operation. Note also that C is case sensitive. Thus, the name TimeStatus is different from timestatus or TIMESTATUS. The #include and #define words are used by the C preprocessor. The former indicates that source code is to be included from a library, while the latter performs a global replacement of the first string by the second string. The line numbers are added by the compiler on the listing only. The functions beginning sys$... are calls to the VMS operating system and are not a part of C.

Listing 2.

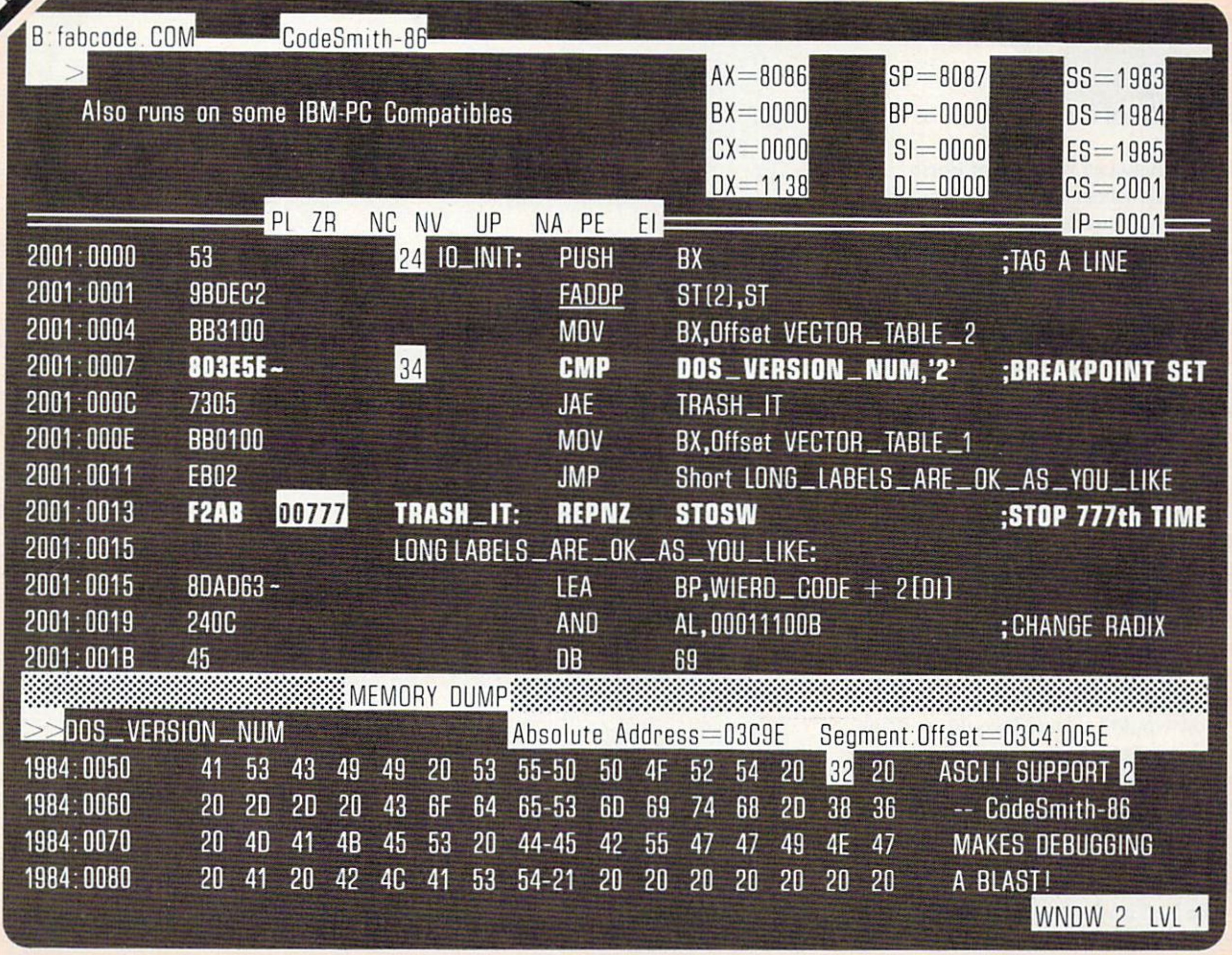
```
                    (Pseudo-code description)

Routine alpha_quick_sort (low_index, high_index)

if low_index = high_index then return endif

if low_index = prev(high_index) then

  call sort2(low_index, high_index)

endif;


partition_index := alpha_partition(low_index, high_index);

call alpha_quick_sort(low_index, partition_index);

call alpha_quick_sort( next(partition_index), high_index) );

return;

                         (Forth program)

Screen # 10001

  : alpha-quick-sort ( low-addr high-addr -- : Sort strings )

    over over >=               ( are addresses equal? )

    if drop drop exit then     ( don't sort a singleton )

```

Listing 3.

IBM-PC SYMBOLIC DEBUGGER

New Release 1.8 – SOLID GOLD

CodeSmith™–86



```
B:fabcode.COM          CodeSmith-86
  >
                                                AX=8086     SP=8087     SS=1983
     Also runs on some IBM-PC Compatibles       BX=0000     BP=0000     DS=1984
                                                CX=0000     SI=0000     ES=1985
                                                DX=1138     DI=0000     CS=2001
               PL  ZR   NC  NV   UP   NA  PE  EI                        IP=0001
2001:0000   53                24 IO_INIT:  PUSH    BX                       ;TAG A LINE
2001:0001   9BDEC2                         FADDP   ST(2),ST
2001:0004   BB3100                         MOV     BX,Offset VECTOR_TABLE_2
2001:0007   803E5E-           34           CMP     DOS_VERSION_NUM,'2'      ;BREAKPOINT SET
2001:000C   7305                           JAE     TRASH_IT
2001:000E   BB0100                         MOV     BX,Offset VECTOR_TABLE_1
2001:0011   EB02                           JMP     Short LONG_LABELS_ARE_OK_AS_YOU_LIKE
2001:0013   F2AB     00777    TRASH_IT:    REPNZ   STOSW                    ;STOP 777th TIME
2001:0015                     LONG LABELS_ARE_OK_AS_YOU_LIKE:
2001:0015   8DAD63-                        LEA     BP,WIERD_CODE + 2[DI]
2001:0019   240C                           AND     AL,00011100B             ;CHANGE RADIX
2001:001B   45                             DB      69
                          MEMORY DUMP
>>DOS_VERSION_NUM                   Absolute Address=03C9E   Segment:Offset=03C4:005E
1984:0050    41 53 43 49 49 20 53 55-50 50 4F 52 54 20 32 20    ASCII SUPPORT 2
1984:0060    20 2D 2D 20 43 6F 64 65-53 6D 69 74 68 2D 38 36     -- CodeSmith-86
1984:0070    20 4D 41 4B 45 53 20 44-45 42 55 47 47 49 4E 47     MAKES DEBUGGING
1984:0080    20 41 20 42 4C 41 53 54-21 20 20 20 20 20 20 20     A BLAST!
                                                                      WNDW 2  LVL 1
```

It's here—THE Multi-Window Interactive Debugger that's STATE-OF-THE-ART.

- Scroll Up/Down thru full-screen disassemblies & memory dumps
- **Load** and **Write** Commands much easier, more powerful than DEBUG's
- "Snapshot" a complete debugging state onto disk—resume later
- True passpoints and execution path counters
- SCREENSAVE mode saves and restores user's graphic display when breakpoint hit
- Disassemble selected ranges of memory code to disk—compatible with IBM Assembler
- Stop on data Read/Write or memory range access

Hot-Line technical support

The Professional's Choice—CodeSmith-86

Multiple copies purchased by:
Lotus Development Corp., MicroPro, VisiCorp, IBM.

Requires MS-DOS & 160K RAM.

OEM and dealer inquiries invited.

VISUAL AGE
642 N. Larchmont Blvd. • Los Angeles, CA 90004

CIRCLE 54 ON READER SERVICE CARD

CodeSmith, TM International Arrangements, Inc.
MS, TM Microsoft Corp.
IBM, TM International Business Machines Corp.

Send ___ **VISUAL AGE** • 642 N. Larchmont Blvd. L.A., CA 90004 • (213) 439-2414
CodeSmith-86 @ $145 + $4 shipping (each) • California residents add 6.5% sales tax.

Name ______________________
Company ______________________
Address ______________________
City ______________ State ______ Zip ______ Phone # ______
Acct. No. ______________________
Exp. Date ______________________
☐ VISA ☐ MasterCard
☐ Payment Enclosed
☐ COD

CALL (213) 439-2414
$145.00

ferences between versions of Forth.

The disk access scheme is a pain. It's been called "virtual storage," but it's more like "cache with least-recently-used replacement." The lack of a standard filing system and communications protocol makes it difficult to transmit data from one system to another. The screen limitation makes it inconvenient to write readable code.

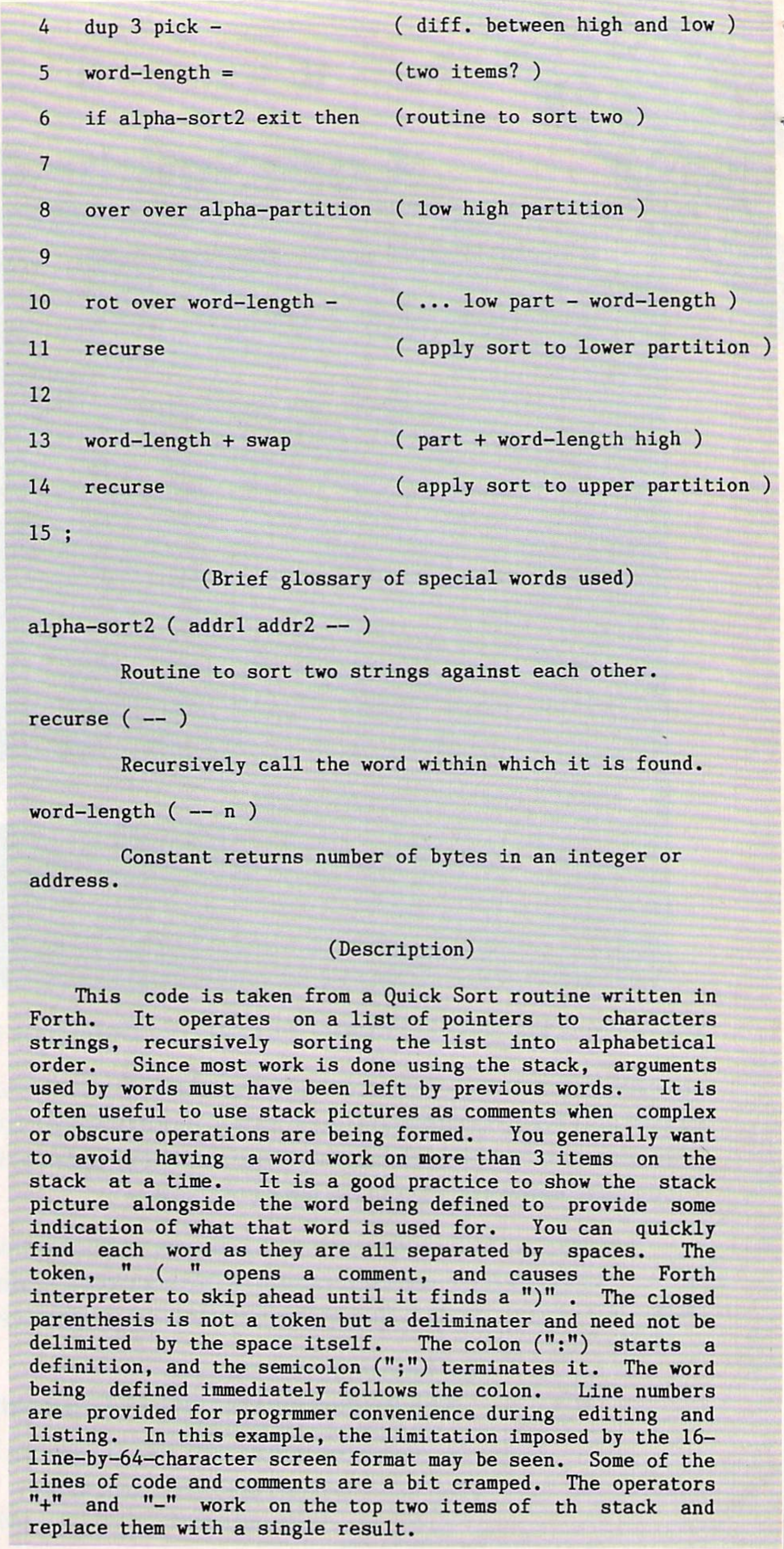
For real-time work, for which Forth is best suited, a priority-encoded multitasking system would be desirable.

So, given all this, which language is better? Which should you use? On the one hand, C provides you with a fast, compiled language, a wealth of utilities to draw upon, and a standard, familiar syntax. On the other hand, Forth gives you total, immediate control of the machine and complete extensibility.

How would I select? I would try using both languages, learn them both, and try to use them for some simple tasks. Then, in the time-honored tradition of computer science, I would arbitrarily elect to use the one that appealed to me most at the time.

The best way to learn any language is by using it. However, if you wish to read more about C, there are numerous books available, starting with *The C Programming Language* by Brian Kernighan and Dennis Ritchie, published by Prentice-Hall (ISBN 0-13-110163-3). The standard for Forth programmers is *Starting FORTH* by Leo Brodie, published by Prentice-Hall (ISBN 0-13-842922-7).

I truly look forward to being able to provide a well-moderated debate column every month in *COMPUTER LANGUAGE*. And I hope you will write in to me with your feedback on this and future debates. Leave me a message on the bulletin board service or send a letter to: Designer's Debate, *COMPUTER LANGUAGE*, 131 Townsend St., San Francisco, Calif. 94107. ■



```
 4  dup 3 pick -                ( diff. between high and low )

 5  word-length =               (two items? )

 6  if alpha-sort2 exit then    (routine to sort two )

 7

 8  over over alpha-partition   ( low high partition )

 9

10  rot over word-length -      ( ... low part - word-length )

11  recurse                     ( apply sort to lower partition )

12

13  word-length + swap          ( part + word-length high )

14  recurse                     ( apply sort to upper partition )

15 ;
```

(Brief glossary of special words used)

`alpha-sort2 ( addr1 addr2 -- )`

Routine to sort two strings against each other.

`recurse ( -- )`

Recursively call the word within which it is found.

`word-length ( -- n )`

Constant returns number of bytes in an integer or address.

(Description)

This code is taken from a Quick Sort routine written in Forth. It operates on a list of pointers to characters strings, recursively sorting the list into alphabetical order. Since most work is done using the stack, arguments used by words must have been left by previous words. It is often useful to use stack pictures as comments when complex or obscure operations are being formed. You generally want to avoid having a word work on more than 3 items on the stack at a time. It is a good practice to show the stack picture alongside the word being defined to provide some indication of what that word is used for. You can quickly find each word as they are all separated by spaces. The token, " (" opens a comment, and causes the Forth interpreter to skip ahead until it finds a ")" . The closed parenthesis is not a token but a deliminater and need not be delimited by the space itself. The colon (":") starts a definition, and the semicolon (";") terminates it. The word being defined immediately follows the colon. Line numbers are provided for progrmmer convenience during editing and listing. In this example, the limitation imposed by the 16-line-by-64-character screen format may be seen. Some of the lines of code and comments are a bit cramped. The operators "+" and "-" work on the top two items of th stack and replace them with a single result.

Listing 3.

CONFUSED?

Communications Software Can Be a Real Headache. For FAST RELIEF, use COMMX! It's Simple to Operate and Provides the Best Features Available for Both Personal and Business Communications:

- Easy to Use Menu Selections and Prompts
- Auto-Dial-Logon and Unattended Controls
- Dial Directory Handles up to 700 entries
- Install Utility for Intelligent Modems
- Programmable Terminal Emulation!
- Linkup with Information Services like WU Telex, TWX, USPS ECOM, CompuServ, NewsNet (free subscription included)
- Micro to Micro and Micro to Mainframe multiple File Transfer Protocols:
  - Text Upload/Download with Options
  - Text and Binary Upload/Download with proprietary Error-free COMMX protocol mainframe Versions available for VAX, CompuServe, DEC 10, IBM 370, HP3000, PRIME
  - MODEM7 Batch and Single file Send/Recv
- Direct Link High Speed Data Transfers
- Electronic Mail Management Software upgrade Available for Organizations
- InfoWorld Report Card A + + + + Dec 1981

COMMX is priced from $195 (micro CP/M or MS-DOS) to $900 (mainframe). OEM and multiple licenses available.

HAWKEYE GRAFIX Inc ™

818-348-7909 / 213-634-0733
23914 Mobile, Canoga Park, CA 91307

CIRCLE 24 ON READER SERVICE CARD

MicroMotion

MasterFORTH

It's here — the next generation of MicroMotion Forth.

- Meets all provisions, extensions and experimental proposals of the FORTH-83 International Standard.
- Uses the host operating system file structure (APPLE DOS 3.3 & CP/M 2.x).
- Built-in micro-assembler with numeric local labels.
- A full screen editor is provided which includes 16 x 64 format, can push & pop more than one line, user definable controls, upper/lower case keyboard entry, A COPY utility moves screens within & between lines, line stack, redefinable control keys, and search & replace commands.
- Includes all file primitives described in Kernigan and Plauger's Software Tools.
- The input and output streams are fully redirectable.
- The editor, assembler and screen copy utilities are provided as relocatable object modules. They are brought into the dictionary on demand and may be released with a single command.
- Many key nucleus commands are vectored. Error handling, number parsing, keyboard translation and so on can be redefined as needed by user programs. They are automatically returned to their previous definitions when the program is forgotten.
- The string-handling package is the finest and most complete available.
- A listing of the nucleus is provided as part of the documentation.
- The language implementation exactly matches the one described in FORTH TOOLS, by Anderson & Tracy. This 200 page tutorial and reference manual is included with MasterFORTH.
- Floating Point & HIRES options available.
- Available for APPLE II/II+/IIe & CP/M 2.x users.
- MasterFORTH – $100.00. FP & HIRES – $40.00 each
- Publications
  - FORTH TOOLS – $20.00
  - 83 International Standard – $15.00
  - FORTH-83 Source Listing 6502, 8080, 8086 – $20.00 each.

Contact:

MicroMotion
12077 Wilshire Blvd., Ste. 506
Los Angeles, CA 90025
(213) 821-4340

CIRCLE 33 ON READER SERVICE CARD

BASIC Becomes a Structured Language

The founders discuss the language's past, present, and future.

By John G. Kemeny, Thomas E. Kurtz, and Brig Elliott

BASIC was born at 4 a.m. on May 1, 1964 at Dartmouth College. In the 20 years since then, it has probably become the most widely used programming language in the world. But during this time, BASIC has undergone many transformations—and not all of them have met with the approval of the original designers.

We invented BASIC for use in schools since students needed a simple but powerful computer language. Evidently a great many other people also wanted such a language, and BASIC grew in popularity in the 1970s.

But the language we designed 20 years ago, though good for its day, is no longer acceptable.

Commercial versions of BASIC have remained frozen at about the 1964 level of Dartmouth BASIC. At Dartmouth, however, our own BASIC grew and evolved as knowledge of programming languages evolved and as programming problems grew more complicated. We added subprograms to BASIC in the late 1960s, graphics in the early 1970s, and good control structures a bit later in the mid-1970s.

But we watched with increasing dismay as more and more students outside of Dartmouth grew up on bad imitations of our 1964 BASIC. So we decided to take matters into our own hands.

This article describes a new implementation of BASIC called True BASIC™. It's easy to learn and easy to use, in keeping with one of the original goals of BASIC. But it also includes developments from the past 20 years of computing. For instance, True BASIC provides multi-line functions and subprograms (with parameters, both internal and external), matrix manipulation statements, several kinds of files, fancy graphics, and more. Programs written in True BASIC will run virtually unchanged on most of the popular new microcomputers, such as the IBM PC and the Apple Macintosh.

Future BASICs will increasingly be based on the ANSI Standard, and they will all have the same features: functions, subroutines, matrix statements, fancy graphics, and so on. True BASIC is based on the ANS draft standard for BASIC,[1] now in the late stages of its technical development.

If that were all one could say about True BASIC, the article would end right here. True BASIC is interesting for two other reasons: it is built around a simple user interface based on a flexible screen editor, and its implementation is especially designed for educational settings. In both cases, we have tried to duplicate onto microcomputers the Dartmouth computing environment we have known and loved. We first describe the language.

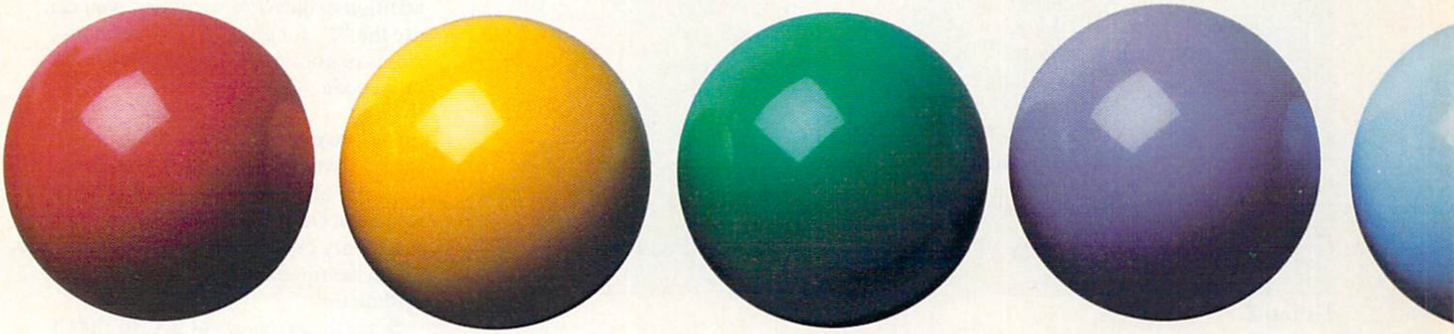


Photo by Dow Clement Photography

A structured language

In 1964, the term "structured programming" had not yet been uttered. The majority of programs were written in assembly language. Then came COBOL and FORTRAN. And the *GOTO* statement reigned supreme. But for at least 10 years, BASIC at Dartmouth has included more sophisticated control structures such as *do-while*, *if-then-else*, and *select-case*.

What do these control structures mean to a programmer? Well, suppose you want to write a program that flips coins, prints whether each coin is heads or tails, and keeps track of the results. Then, after flipping 50 coins, it reports the total number of heads and tails.

Written in Old BASIC (which is close to the 1964 version), the resulting program is seen in Listing 1. The program loops 50 times, and each time it checks to see if a random number was less than one-half. If so, it then counts it as a tail. If not, it counts it as a head.

```
100 REM FLIP COINS AND KEEP TRACK OF RESULTS.
110 REM
120 FOR I = 1 TO 50
130    IF RND < .5 THEN 170
140    PRINT "Heads."
150    LET H = H + 1
160    GOTO 190
170    PRINT "Tails."
180    LET T = T + 1
190 NEXT I
200 PRINT "Heads = "; H; "Tails ="; T
210 END
```

Listing 1.

```
!  Flip coins and keep track of results
!
for i = 1 to 50
    if rnd < .5 then
       print "Tails."
       let tails = tails + 1
    else
       print "Heads."
       let heads = heads + 1
    end if
next i
print "Heads ="; heads; "Tails ="; tails
end
```

Listing 2.

```
!  Print values of Sin between 0 and 20.
!
for x = 0 to 20 step .2

   print x, Sin(x)

next x
end
```

Listing 3.

The heart of this problem is the *if-then* test. Let's see how this works out in True BASIC (Listing 2). True BASIC uses an *if-then-else* structure to express the two-way branch. The result is the same, but isn't the second program easier to read than the first?

True BASIC doesn't require line numbers unless you use *GOTO*-style statements. Line numbers are never really necessary since True BASIC supplies all the constructs needed for structured (*GOTO*-less) programming, and several more. These include: *select-case* with several ways to express each case; *do-while* and its counterpart *loop-until*, an exception trapping structure; and structured ways to escape from loops, functions, and subroutines.

Also, True BASIC doesn't insist on using capital letters. Uppercase-only terminals are a vanishing breed. Use capital or small letters as you please. And, in addition to the *REM* statement, you can use the "!" for either *REM*-type comments or comments at the end of other code.

Easy graphics

Back in 1964, few outside of research laboratories had the machinery to produce graphical output. Anyone who had such machinery could be expected to count pixels on the screen and do all their own arithmetic.

Since the inventors of BASIC didn't have such machines, the original BASIC did not have graphics. When interactive graphics did appear in the late 1960s, students at Dartmouth could draw pictures without counting pixels. And yet, when microcomputers became popular in the mid to late 1970s, their BASICs still required you to count pixels and do little sums in your head.

But True BASIC makes things easier. Just as the original BASIC made printing numbers and strings simple—no complicated format lists—so True BASIC makes drawing pictures simple.

Let's look at a True BASIC program that prints the values of the sine function between zero and 20. It's a simple program (see Listing 3). The result describes a sine wave but not in a very convenient form. It prints a long table of numbers, part of which is seen in Table 1. A small modification makes everything much clearer. Change the *PRINT* statement to a *PLOT LINES* statement. Add a *SET WINDOW* command to describe what portion of the coordinate plane to view (see Listing 4).

```
0                    0
.2                   .198669
.4                   .389418
.6                   .564642
.8                   .717356
1                    .841471
1.2                  .932039
1.4                  .98545
1.6                  .999574
1.8                  .973848
2                    .909297
```

Table 1.

In True BASIC, you describe how to plot things in your terms, not the computer's. One *SET WINDOW* statement can set your coordinates to run from zero to 20 (on the *x*-axis) and -2 to 2 (on the *y*-axis). True BASIC then does the messy work of figuring out pixels for you. If you move your program unchanged to another computer with a different number of pixels on the screen, it will still draw the same picture.

More sophisticated graphics let you define pictures, which you can then use like stencils. A picture is a graphical subroutine that can be drawn with (or without) various two-dimensional transformations applied to it. You can enlarge or shrink them, move them around, rotate them, or shear them, alone or in combination.

You may also fill areas of the screen with colors, cut and paste sections of the screen, introduce animation, and so forth. Lack of space prevents a complete description, but True BASIC includes the proposed ANSI BASIC graphics, which in turn are based on the international GKS level Ob standard.

Recursion

Functions that can call themselves are a popular and important part of computer science. The success of the LOGO language attests to that. Along with many of our colleagues in the early days, we paid no attention to this important idea. But we now realize that recursion provides the only sensible way for looking at certain problems in computer graphics, list manipulation, and equation solving.

True BASIC lets you define your own functions, and naturally they may call themselves. A simple example is the factorial function. The factorial of a number *n*, Fact(*n*) is defined as:

```
Fact(n) = n * (n-1) * (n-2) * ... * 1
```

That is, Fact(5) = 5*4*3*2*1, or 120. But since

```
Fact(n-1) = (n-1) * (n-2) * ... * 1
```

you can combine the two equations as follows:

```
Fact(n) = n * Fact(n-1)
```

provided that you also make a special check when $n = 1$, to make the chain of calls eventually stop. (Otherwise, we'd try to compute Fact(l) = 1 * Fact(0), and so on, with no end in sight.)

You can easily write Fact in True BASIC (Listing 5). Such a recursive program can be written in the 1964 version of BASIC, but it's very messy. It makes such a puzzle that readers of *COMPUTER LANGUAGE* may wish to try to figure it out. Unfortunately, this is the simplest way that recursion can be achieved in many current versions of BASIC (Listing 6).

Of course, most sensible programmers would calculate factorials using a loop rather than recursion. But there are many common problems, such as the computer solution to the Tower of Hanoi, which are simple with recursion and ridiculously complicated without it. In Old BASIC the best one could do to "fake" recursion is illustrated in the previous example.

The user interface

The second major component of the True BASIC system is the user interface. It continues the well-established model used at Dartmouth for over 20 years, which is also followed by most current micro BASICs. It has these features:

- Commands that look like English words
- An automatic (default) source program editor
- Features based on the notion of a "current file"

To these, True BASIC adds:

- A modern, window-based screen editor

It surprises us that so few writers of computer software (except for business software) realize that most of the users are, and will remain, novices. Familiar words are much easier to remember than computer jargon. Isn't *HELLO* simpler than *LOGIN*, and *GOODBYE* or *BYE* simpler than *LOGOUT*? How about *OLD* versus *LOAD*, and (our all time favorite) *LIST* versus *CATENATE*? Furthermore, a simple command like *RUN* does everything necessary to "run" the program. For example, if the program is in source code form, *RUN* compiles, loads, and executes. If the program has already been compiled, RUN does only what is necessary.

A single command, *OLD*, not only specifies the name of the program you want to modify or run, but also automatically invokes the screen editor. Users need not even be aware that there is a separate program editor. This style is consistent with experience at Dartmouth, except that in the old days the editor used line numbers. (In fact, True BASIC gives you both a screen editor and a line-numbered editor. You can use both at the same time if the program has line numbers.)

The heart of the user interface is the

screen editor. We cannot describe it in detail because its features and style of use depend on the underlying operating system. For instance, on the IBM PC the screen editor is built around cursor controls and function keys, while on the Apple Macintosh it is based on mice and menus.

The user interface is built around windows. There are a minimum of two windows—one for the screen editor and one for input and output. On machines that provide dynamic windows, like the Macintosh, more than two windows can be used. The contents of the windows can be scrolled.

Since the screen editor is an integral part of the system, it can be invoked during the running of a program to, for instance, display erroneous parts of a program. Furthermore, the screen editor can be used separately. It can be used for ordinary word processing by adding a few simple formatting features.

Expert users may prefer a different style of user interface with more direct control over what goes on. They may be willing to remember a bigger list of commands, some of them with mysterious names. But once again we remind the reader that the majority of our users will be novices.

What about the advanced user?

It's no surprise that BASIC is good for programs that are 10 to 100 lines long. But what about programs that are 10 to 100 pages long? Serious software developers need good tools to help write such long programs.

We've always believed that BASIC should be an appropriate language for serious programming. Just because a program is long doesn't mean that it should be hard to write. The language should help you as much with long programs as with short ones. And so True BASIC lets you:

- Use all the available memory on your machine without any complicated overlay or segment instructions
- Write programs or functions as separate units, put them in libraries, compile them separately
- Transport programs from computer to computer with no rewriting
- Debug with True BASIC's built-in debugger
- Create and edit programs, subprograms, or functions with the built-in screen editor
- Take advantage of floating-point processors (e.g. 8087) automatically

The longest True BASIC program written as of April 1984 contains about 150 separately compiled subprograms. Its listing is over 180 pages long. It runs on the IBM PC.

Implementation

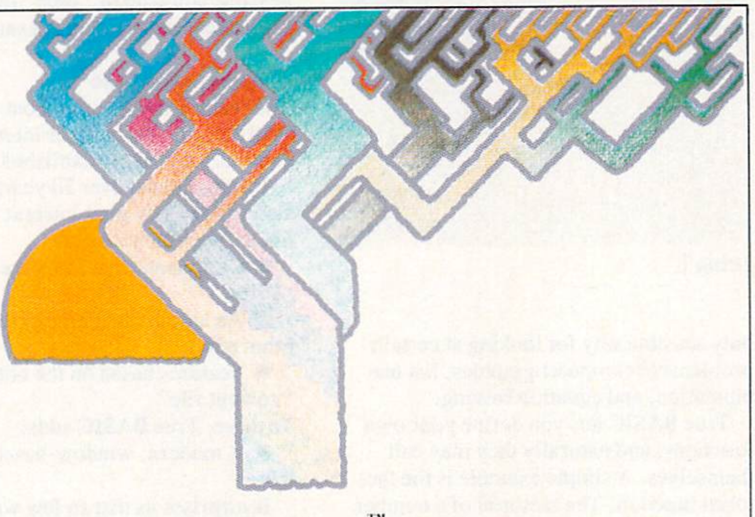
The system is split into four parts. We have already discussed the user interface. The other three are: the compiler, interpreter, and run-time package.

The compiler translates your program into an intermediate code. The interpreter executes this code. The run-time package takes care of complicated I/O statements, built-in functions, and screen management for the three other parts.

The four parts together take up about 64K bytes of memory. Since the entire system resides in memory, one needs at least 128K bytes of memory to run an average-size program. Since True BASIC lets you use all the memory you've installed, you can buy more memory if you need more room. (Making more than 64K bytes of memory accessible on the Intel 8088 processor was difficult, but we think it's a shame not to use all the memory that is actually there.)

This style of compiler/interpreter combination was first made popular by the Pascal P4 compiler from ETH in Zurich, Switzerland.[2] The P4 system had only a compiler and an interpreter, but both were written in Pascal itself.

The result was impressive. Pascal quickly became a popular language, in large measure because it proved so easy to transport the P4 system to new computers. Instead of rewriting the entire language system for a new machine, the



Btrieve.™

Network and single-user file management for all your programs. All your programming languages.

Say goodbye to writing file management routines. Because now Btrieve™ can handle file management for all your program development on the IBM PC. All your programming languages.

Based on the b-tree file indexing system, Btrieve provides the most sophisticated file handling powers available for your PC. And for multiple PCs. Btrieve™/N lets you share files among PCs in NetWare,™ PCnet,™ EtherSeries,™ or MultiLink™ networks. And both Btrieve and Btrieve/N offer the same superior performance characteristics:

- Interfaces all major IBM PC languages—BASIC, Pascal, COBOL, C, and IBM Macro Assembler
- Written in 8088 Assembler for the IBM PC
- Multikey access to records
- Automatic file recovery on system crash.

Btrieve and Btrieve/N also have user-defined transaction management. Unlimited number of records per file. Complete error control and recovery within an application. Duplicate, modifiable, and segmented keys. Variable cache buffer from 16K bytes to 64K bytes.

Moreover, record retrieval is *fast* with Btrieve—no matter how large your data base. So *you* can begin writing programs *faster* with Btrieve.

Say goodbye to file management routines. And hello to Btrieve.

SC SoftCraft Inc.

P. O. Box 9802 #590 Austin, Texas 78766 (512) 346-8380

Suggested retail prices: Btrieve, $245; Btrieve/N, $595. Requires PC/DOS or MS™/DOS, version 1 or 2. Dealer inquiries welcome.

IBM, MS, Btrieve and Btrieve/N, PCnet, MultiLink, NetWare, and EtherSeries are trademarks of International Business Machines, Microsoft Corporation, SoftCraft Inc., Orchid Technology, Davong Systems Inc., Novell Data Systems, and 3Com Corp., respectively.

CIRCLE 50 ON READER SERVICE CARD

```
!  Plot values of Sin between 0 and 20.
!
set window 0, 20, -2, 2

for x = 0 to 20 step .2

   plot lines: x, Sin(x);

next x
end
```

Listing 4.

implementors needed only to write a new interpreter. When that worked, the system was up and running.

The P4 system compiled Pascal into p-code. True BASIC compiles into b-code. The differences between b-code and p-code are subtle, though we believe we've made some improvements. But we tip our hats to the P4 team, who were the first to make it all work on a large scale.

Why not a pure interpreter?

Some readers may wonder why we chose a compiler-based system when our target market is education. Doesn't current wisdom mandate an interpreter for novices? Don't error messages (about syntax errors) have to appear fast? Aren't compilers for experts?

Some feel that one needs an interpreter if one wishes to keep the user interface simple and see the error messages quickly. Contrast this with the fact that Dartmouth BASIC has always been compiled even when machines were slow and even back in 1964.[3] What are the reasons?

First, about error messages. We disagree with the practice of those small BASIC interpreters that provide line-by-line syntax error detection while the user is entering the program. Such local error detection completely fails to identify global errors, such as the *NEXT* failing to match the *FOR*. Also, such immediate error detection interferes with the program-entering stage, especially if the user is a good typist. What truly is important is that the error messages are displayed quickly. And this is what a fast, single-pass, load-and-go compiler can do.

Compiler

The compiler is highly optimized for both space and speed. Since it is memory resident, it can't take up too much room. Since it must compile the source program every time it's run, it must be quite fast. These sound like difficult goals but, in fact, people have known how to write small, fast compilers since the early 1960s.

As we asserted, Dartmouth BASIC has always been compiled, even in 1964. Since the principles of compiler construction (and, in particular, of symbol table management) were only then being discovered, the earliest BASIC was simplified in certain ways to make compiling easier. Thus it allowed only one one- or two-character variable names.[3]

On the other hand, the resulting machine code was very speedy—much faster than an interpreter would have been—even though the code was never optimized. For instance, the first compiler created one line-number table. When a *GOTO* statement was executed it jumped indirectly through this table. Many of today's commercially available BASICs haven't learned this trick; they scan the entire program from the start to find the target line number.

Today's compiler technology is a little more advanced. The True BASIC compiler keeps track of jumps itself and emits b-code that contains direct jump instructions. It uses a heavily modified Samelson-Bauer[4] bottom-up parser, which is very speedy. The compiler is an "in-memory" compiler, so it reads the source code from memory and places the b-code directly in memory. Since no disk access is required, compiling is fast.

Interpreter

At the heart of the True BASIC system is the interpreter. It's written in assembly language and executes the b-code produced by the compiler. If the interpreter is slow, the entire system will run slowly. Therefore, a great deal of care has been expended on True BASIC's interpreters.

Naturally, the interpreter must be rewritten for each new processor supported. The very first True BASIC interpreter was written in C for the Z8000 processor. The prototype True BASIC system runs on an IBM PC, so its interpreter is written for the Intel 8088 processor in assembly language. The same holds true for the next target machine, the Apple Macintosh, which has a Motorola 68000 processor. Each new processor requires a new interpreter, which must be rewritten from scratch.

But having identical processors is not always enough. Some changes to the interpreters may still be required. For instance, the IBM PC, IBM PCjr, and DEC

In 1964, under the direction of John Kemeny and Thomas Kurtz at Dartmouth College, the BASIC computing language was born. As an acronym for "Beginner's All-purpose Symbolic Instruction Code," BASIC was first implemented on the GE 225 computer. What made BASIC so different from the mainframe languages available during that time was that there suddenly became a way for the person with little or no experience in computers or mathematics to learn programming easily. Kemeny and Kurtz also created one of the first widely-used timesharing systems which has been adopted by several well-known commercial systems.

As other colleges and computer manufacturers adopted BASIC, they added enhancements to meet their own particular needs. Out of this came Extended BASIC, SUPERBASIC, XBASIC, BASIC PLUS and others. Although a standard was developed in 1978, a wide variety of BASICs still exist with many similarities, but with many individual quirks as well.

The most widely used version of BASIC in the microcomputer field was developed by Microsoft and is usually referred to as MBASIC. Although these BASICS are availible on a number of microcomputers, the language is implemented differently on each system.

As Chairman of Dartmouth's Mathematics Department for twelve years, John Kemeny also served as President of Dartmouth College from 1970 to 1981. In 1979, he took time out to act as Chairman of the President's Commission on the Accident at Three Mile Island. He has authored roughly 100 publications, including many influential books and articles on mathematics and computing.

Thomas Kurtz was director of the Kiewit Computation Center at Dartmouth from 1966 to 1975. He now serves as Chairman of Dartmouth's graduate program in Computer and Information Science. From 1974 through 1984, Dr. Kurtz served as Chairman of the American National Standards committee X3J2, charged with developing a standard for BASIC.

—by Hugh Byrne

ACTIVE TRACE

"A marvelous Basic programming aid . . .

It's just amazing to watch a program you wrote run under Scope, and debugging becomes if not trivial, then at least doable."

Thomas Bonoma, *Microcomputing*
Dec. '83, p 22

"Extremely useful program . . . Anyone doing much programming in Basic should appreciate Active Trace a lot."

Jerry Pournelle, *Byte Magazine*
April '83, p 234

Spaghetti code is what many "experts" call a beginner's Basic program which is all tangled up and difficult to follow. The **Active Trace** package will help you learn how to avoid the pitfalls of structureless programs. And if you already have a program which is too confusing to follow, or has an error which is hiding, relax. **Active Trace** doesn't get confused. **Active Trace** will lead you through your program letting you know variable values (all variables or just those you specify) as they change. In a form a novice can understand, your program's internal activity is presented on your screen, or printer, or it can be saved on disk.

For more advanced programmers, the disk file of your programs activity can be used with your word processor to automatically find the source of an error and display the circumstances surrounding its occurrence.

Ready to Order?
Just have a Question?
Contact your dealer or call
Toll Free: 800-358-9120(US)
800-862-4948(CA)

Active Trace $79.95
Complete package includes Scope, XREF mapping and documentation

Scope Separately $49.95
Only recommended for those who already own professional mapping (XREF) programs

Active Trace is available for most **MS-DOS** and **CPM 2.2** systems and supports the special features of Brand specific versions of Microsoft Basic such as **Basica** on the **IBM-PC**.

AWARECO
Active Software
P.O. Box 695 Gualala, CA 95445
(707) 884-4019

Active trace, Active software, AWARECO and Scope are trademarks of A Ware Company—CPM is a trademark of Digital Research—MS-DOS and Microsoft are trademarks of Microsoft Corporation—IBM-PC is a trademark of IBM Corp.

CIRCLE 2 ON READER SERVICE CARD

Rainbow all have the same 8088 processor. However, they differ in other ways. Graphic implementations usually cause the most problems since each computer has its own way of drawing lines on the screen, switching colors, and so forth.

To make things worse, even identical computers that run different operating systems may require interpreter changes. For example, the instructions used to save a file are different under the PC-DOS and CP/M operating systems, even when running the same computer. True BASIC uses the native operating system for the machine, rather than having its own customized operating system. (On the IBM PC, the preferred operating system is PC-DOS.) A different interpreter would be needed in order to run under CP/M. On the Macintosh, it will run in the standard windowing environment.

Much design effort has been focused on making the interpreter simple. The less the interpreter has to do, the less assembly code must be changed when transporting the system to a new machine.

BASIC history

More details about the history of BASIC can be found in the chapter "BASIC session" in *History of Programming Languages*[3] and in *Back to BASIC*.[5] Further details of BASIC are described in "Standard BASIC—On its way,"[6] and "True BASIC."[7]

References

1. "Proposed Draft American National Standard for BASIC," Document X3J2/84-10 (revised versions will appear), X3 Secretariat, CBEMA, 311 First Street N.W., Suite 500, Washington, D.C. 20001.
2. Nori, K. V., et al, "The Pascal (P) Implementation Notes," ETH, Zurich. 1975.
3. Kurtz, T.E., "Basic Session", in Wexelblat, R.L. (ed.), *History of Programming Languages*, (Academic Press, 1981): 515-549.
4. Bauer, F.L., and K. Samelson, "Sequential formula translation," CACM 3, no.2, (Feb. 1960): 76-833.
5. Kemeny, J.G., and Kurtz, T.E., *Back to BASIC*, (Addison-Wesley, 1984).
6. Kurtz, T.E., "Standard BASIC—On Its Way", *BYTE* vol. 7, no. 6, (June 1982): 182.
7. Elliott, Brig, "True BASIC", *BYTE* vol.9, no.4, (April 1984): 300.

```
! Compute factorials of numbers 1 to 10.
!
def Fact(n)
    if n=1 then let Fact=1 else let Fact=n*Fact(n-1)
end def

for i=1 to 10
    print i, Fact(i)
next i
end
```

Listing 5.

```
100 REM  COMPUTE FACTORIALS OF NUMBERS 1 TO 10.
110 REM
120 FOR J = 1 TO 10
130     LET N = J
140     GOSUB 1000
150     PRINT J, F
160 NEXT J
170 STOP
1000 REM  FACTORIAL SUBROUTINE
1010 IF N > 1 THEN 1040
1020     LET F = 1
1030     RETURN
1040 LET D = D + 1
1050 LET S(D) = N
1060 LET N = N - 1
1070 GOSUB 1000
1080 LET F = F * S(D)
1090 LET D = D - 1
1100 RETURN
1110 END
```

Listing 6.

Programming in the UNIX Environment

By Bruce H. Hunter

Certain operating systems have been traditionally associated with certain programming languages. When you think of an IBM operating system like OS/VOS, DOS/VSE, MVS, or VM, you tend to automatically associate it with languages like 360 assembly, JCL, and PL/1. When you think of small micro operating systems, you think of resident BASIC.

Until 1983, CP/M was the home of 8080 assembly and a great number of compilers for languages such as MBASIC, CBASIC, Pascal MT+ and others. With the influence of UNIX spreading across the micro-to-maxi spectrum of computers, the languages associated with UNIX have been receiving a lot more attention. Some of the UNIX languages are already familiar under other operating systems, but some of them are brand new to all non-UNIX programming environments.

The primary language of UNIX is C, but there are many other UNIX languages. RATFOR (rational FORTRAN) is somewhat of an anomaly now, but it used to be a UNIX language favorite. Lex, the lexical analyzer, and yacc, the UNIX compiler compiler, are both legitimate UNIX languages. The Shell interpreters are languages in their own right and include the Bourne and C Shells. Awk is both a language and a filter. Of the languages similar to those in other operating systems, UNIX also includes bas, a BASIC dialect, and f77, FORTRAN ANSI 77. Three of the so-called "text processors" (roff, nroff and troff) are actually more like languages. Depending on the version of UNIX and the type of installation, there may be many other languages to be found, including PL/1, APL, Ada, Pascal and COBOL. First let's look at C.

The C programming language

It is not easy to accurately and comprehensively describe C in a single article. It has been called a high-level assembler, and that description is a good beginning. C is a structured language that is entirely function oriented. The program main is a function, and all called subroutines and system calls are functions as well. C functions do not have to receive parameters, nor do they have to return a value, so even procedures are functions in C.

In spite of the fact that C is a function-oriented language, it is ironic that C under UNIX has few functions of its own. A C function is actually a UNIX subroutine or system call. Consequently, UNIX C is quite different than C compilers that operate on other operating systems. UNIX C calls commands, subroutines and system calls with no hardware dependency. Also, it gives you direct access to the operating system. There is no third party you have to go through to get your hands on the system.

Outside of UNIX, C must make system calls to the operating system via DOS, BDOS or OS calls. This, however, is made transparent to the user because C compilers operating outside of UNIX deal with groups of functions that emulate UNIX operations, similar in a sense to languages that emulate the 8087 math processor. There are many good C compilers that run outside of UNIX, including some fine ones for 8- and 16-bit micros. Now that UNIX software is beginning to emerge, we are even beginning to see enhanced UNIX C compilers such as Safe C by Alan Feuer, author of *The C Puzzle Book*, published by Prentice-Hall.

Like most operating systems created before the last decade, UNIX was originally written in assembly. UNIX was then written in a succession of languages, and today most of UNIX is written in C. Only 10% of the UNIX kernel, the heart of the system, is written in assembly. C was created to replace assembly for systems level work, and it is superb for that purpose. Besides making systems-level programming much faster, C also eliminates most of the hardware dependency associated with assembly.

Because of C's intrinsic hardware independence, it has gained a reputation for being the language of portability. One of the reasons UNIX can be transported without much difficulty is because so much of it is written in C. Proof of this fact is a company called UniSoft that transports UNIX in as short a time as two weeks, for $20,000.

C is becoming so widespread that companies like Digital Research and Gifford Computer Systems have made C their in-house systems language, and this is a continuing trend. Digital Research is in the process of transporting UNIX to Intel's 286 processor, and because many Digital Research languages have been written in C, their language family will be transported as well.

C is the basic language of UNIX, and in many respects it is also the common denominator of UNIX. Several UNIX languages generate C output, including yacc, pc Pascal and f77 FORTRAN. However, C is only part of the UNIX story. Commands that are newly created are usually written in UNIX Shell script and so are system commands that are seldom used. However, once a command is accepted and put in common usage, it is rewritten in C. C code is faster, but Shell script takes up less storage room.

Programming in C is an adventure. C has a thorough list of operators and many data types to choose from. Its programming range is quite large because it is capable of both high-level and low-level programming applications. C's roots are in Algol, so it has a high-level, structured language approach to programming. However, it was created to perform low-level systems work such as the manipulation of system peripherals.

I like to call C a "mid-level" language. Its forte is a high-level programming approach to low-level programming tasks. C code is cryptic, so code blocks of a half-dozen lines in other languages become one-liners in C. It generates tightly optimized code, so C programs are fast and efficient. Most important, the C language is easily enhanced. If it doesn't have the operators or functions you need, it provides you with plenty of programming tools to create your own.

Initially C can be a difficult language to learn—not so much because of the syntax, the symbols, or even its cryptic, shorthand approach to programming, but because C demands an understanding of systems as well as applications programming techniques. You can only use the full potential of C's programming power by understanding what happens at the systems level; so if your programming experience is limited to high-level languages and applications, you'll need to concentrate on learning what systems-level programming is all about. On the other hand, if you've done most of your programming in assembly, you will be unfamiliar with many of the subtleties of high-level, structured programming.

Learning C is well worth the extra effort. Once familiarity has been gained, C becomes a joy to use. C allows you the freedom to access almost any part of the system and its peripherals, outside of the kernel. It can go almost anywhere and do almost anything.

On the other hand, for this programming freedom you pay the price of complete responsibility for doing type checking and conversions, and you must be very careful not to send a pointer into the heart of the system. You can get into a great deal of trouble if you send a pointer to the wrong place, so languages like C are a double-edged sword. Yet in the hands of a knowledgeable programmer, C is one of the most powerful programming languages ever written.

C is a unique language. It is exciting because it allows you to do assembly-level programming more quickly and efficiently. It takes the drudgery out of low-level programming, and it introduces you to the subtleties of high-level languages. You are able to do systems-level work from a high-level perspective, and the inner workings of the operating system become much less of a mystery.

The UNIX Shell

Next to C, the UNIX Shell is the most important utility in UNIX because it is the interface between the user and UNIX. The Shell is in a sense an interpreted language, ready to be called from the minute you log on.

You have probably heard about the UNIX Shell, the Bourne Shell, the Berkeley Shell, and the C Shell. With so many names, it might seem a little confusing at first. Actually, only two main Shells are available under UNIX. The most common Shell is the Bourne Shell, sh, the creation of Steven Bourne. The Bourne Shell is the original Version 7 UNIX Shell. Its counterpart is csh, the C shell, part of the Berkeley-enhanced UNIX distribution and therefore also called the Berkeley Shell. Essentially, the C Shell does what the Bourne Shell does, but it uses C syntax.

The primary purpose of the Shell is to be a command line interpreter, interpreting command lines as they are entered from the keyboard. As such, the Shell is the only interface between the user and UNIX. Similar in a few respects to CP/M's CCP, the UNIX Shell has a simpler side, interpreting command lines one at a time and causing the one-liner instructions entered from the console to be acted upon by UNIX as a whole. In the following example, *cp* is the UNIX command to copy a file:

```
cp textfile textfile.bak
```

The Shell will cause textfile to be copied to textfile.bak.

But the Shell has a more complex side, far more complex than CCP, CP/M's SUBMIT, IBM's EXEC, or EXEC2. It is a full-fledged interpreted language as well. It isn't limited to executing a series of simple commands. The Shell reads source code written in Shell script and interprets each line of code the same way that BASIC and dBASE II do, one line at a time. To give you an idea of how sophisticated the UNIX Shell is, it has incremented and non-incremented loops. It also has an *if-then-else*, an *elif* (*else-if*), and even a *case* structure. The Shell language is geared to deal with numbers or strings, and it does a remarkable job dealing with file names and command line arguments.

As mentioned earlier, the Bourne Shell comes standard with UNIX, but it is not the only Shell in town. For the past several years, the University of California at Berkeley has been a haven for UNIX students, programmers and system programmers, and the result is a whole set of enhancements to UNIX, including the C Shell.

As its name implies, the C Shell is a C-like command interpreter. It uses all of the C operators and constructs, and it also has additional, nifty features like remembering the last lines you typed in at the console (called a history). You can create aliases for command names in the Shell, so it is possible for each user to tailor individual commands to his or her own needs.

To a dyed-in-the-wool C programmer, the C Shell seems like an old friend because it incorporates so much of the C language. The Berkeley Shell (C Shell) is extremely popular, and many of its special features may, due to popular demand, become part of the UNIX standards some day. Until that time, both Shells (Berkeley and Bourne) can be stored and used on UNIX systems. Whichever Shell is being used is the UNIX Shell at the time.

The Shell allows a series of commands to be saved and recalled when needed, which saves the drudgery of retyping them. The best way to explain this concept is to present a Shell program and describe what it does.

Consider the following scenario. A system administrator often has a file containing a standard message of the day, which we'll call *motd.* However, he or she often has to create another *motd* file for specific occasions such as notifying users of a scheduled shutdown. The old *motd* file can be renamed, and when it comes time to go back to the original *motd* file, all that is needed is a swap. The following is a UNIX Shell program in Bourne Shell script which performs the swap:

```
cp motd motd.tmp
cp motd1 motd
cp motd.tmp motd1
rm motd.tmp
ls –l motd*
cat motd
```

Although this series of commands is not difficult to create extemporaneously, having done it once, why ever do it again? These Shell commands are entered to a file as a program by using an editor. The program can be given any appropriate name, like *motd.swap*, and is simply invoked by name, like this:

```
# motd.swap
```

You can also use this technique to create commands useful within your own environment from already existing commands. For example, when roaming around the UNIX file hierarchy, it's not unusual to get lost from time to time. From personal experience, I've found it really handy to be able to find out what machine I'm on, what terminal, where I am in the directory, and what is in the directory, including all file attributes. The following Bourne Shell script does just that:

```
: d a Utility for a full listing with extras
vmid
who am i
date
pwd
ls –l
```

The line beginning with a colon is a comment. *vmid* is a command used on machines running UNIX on a virtual operating system. A number of UNIX virtual machines can exist on a single physical machine. In these cases, it's necessary to have a command to tell you which machine you're on. The *who am i* program line yields the logon name associated with your parent Shell. *date* prompts the time and date. *pwd* tells you where you are in your file hierarchy. *ls —l* will generate a long listing of the files in the current directory.

Shell programs allow you to create applications programs with an interpreted language that interacts with the operating system. If you have to create an applications program, doing it in the Shell is the fastest way. Quick and dirty programs are not difficult to create in either Shell and Shell programs take up very little room in memory. If the Shell program is successful but runs too slowly, as interpreters tend to do, the Shell program can be rewritten in C. Transposing a C Shell program into a C program is not a difficult job—the two languages share syntactic similarities.

Yacc and lex

To fully understand UNIX languages, you need to be familiar with yacc. Yacc is an acronym for yet another compiler compiler. Before yacc, language compilers and interpreters had always been written in assembly. Eventually, many people became interested in defining the properties of a language in terms of its grammar. By making a science of defining the methodology of parsing and lexigraphical analysis, it soon became apparent that a language compiler compiler could be constructed. Soon it was the rage for graduate students to write compiler compilers, and that's when yacc was written.

Yacc interfaces with lex, the lexical analyzer, by first performing a lexical analysis of the program input and then allowing the tokens to be parsed in accordance with the rules of standard BNF syntax specifications. In simpler terms, lex defines what the language tokens will be and then creates a series of specifications showing what will be done with the tokens once recognized. Yacc uses C syntactic conventions. If a compiler writer can clearly define the language syntax, the resulting compiler or interpreter produced by lex and yacc will be smaller than one written in assembly and far better than one created in a high-level language. The parsing tables constructed by yacc are much more efficient than the hundreds of lines of code it used to take to accomplish the important task of parsing. The syntax is translated into a series of tables that do the majority of the work. This is the secret of yacc's success. If there are conflicts in the resulting language, it is because of conflicts in the specifications. Language compilers written in yacc provide tightly optimized code.

Awk

UNIX has a number of filters that are simply programs that modify text passed through them. A typical UNIX filter is *sort*, a utility to sort the contents of a file. Other filters like *grep*, *egrep*, and *fgrep* find patterns in text and print them out either to the screen or another file.

The other UNIX filters are more complicated. Picture a file that is too large to be encompassed by an editor. How are you going to deal with it? *sed* is a screen editor that doesn't emcompass the entirety of a file. The file passes through it, allowing files too large for a conventional editor to be edited. Awk is even trickier yet. It reads through a file like *sed* but with a very big difference. Awk has an entire C-like language to go along with it.

Awk is oriented to files, records, and fields. It will process a file by taking each line and performing all sorts of miracles on any or all fields. Awk can swap them, ignore them, total or average them, or execute any operation that can be conceivably programmed into a file handling utility. The constructs of the language give the programmer C-like operators, a *case* structure, an *if-else,* and all the programming tools expected of any language. Although awk is difficult to learn at first, once mastered it will do wonders. It is a filter and a language.

In programmers' terms, awk is a language tailored for a specific purpose: you write programs in it to filter files. This means that you don't have to write file handling routines. No more open, close, read, write, and other file drudgery. Awk is built for all that. When you learn C, you learn the basic language of UNIX, but when you learn awk, you start getting into the subtleties of the philosophy of UNIX.

f77

It is my opinion that f77 FORTRAN was put into UNIX just so there would be a familiar and recognizable language for those used to other operating systems. f77 FORTRAN is bare bones ANSI 77 FORTRAN with no enhancements. To date, it is the only UNIX language in which I have found flaws. Early versions found on UTS, a mainframe version of UNIX, could not handle *type double* because of a parsing error. Version 2.2 (System III) has this corrected. At Bell Labs, f77 probably received more use by the RATFOR preprocessor than from straight FORTRAN programmers. However, it is a substantial improvement over an enhanced FORTRAN IV or ANSI 66.

FORTRAN is hard to beat as a number cruncher. Its intrinsic functions for math are superior to anything this side of PL/1 or APL. The 77 version is a vast improvement over the 66 version in many ways—one is that it will tolerate white space in the source code, which allows it to approximate a structured language. It has a decent *if-then-else*, *end-if*, and *else-if*, so a good programmer can keep *GOTO*s and labels down to a minimum and write close to top-down, structured code. You can even create a *case* structure. Most important of all, scientists and engineers are still being trained in FORTRAN. As long as FORTRAN refuses to die, at least under UNIX it has a good home.

RATFOR is not a compiler. It is a preprocessor to convert RATFOR source code into FORTRAN 77. RATFOR is a language created in C's image. It does not have C's ability to access individual bits, but it is a viable structured language that is more than capable of holding its own on string processing, FORTRAN's weakness. RATFOR was made nearly immortal by Brian Kernighan and P.L. Plauger's *Software Tools*, published by Addison-Wesley. RATFOR saw some noticeable usage in the last decade, but in today's UNIX and non-UNIX world, it takes a back seat to C. If one has to learn a C-like language, why not C itself? The major philosophical thought behind RATFOR was that it could be transported anywhere FORTRAN could be found, and that was just about anywhere in the pre-micro world.

Pascal

Pascal is a borderline UNIX language. It is not part of the original AT&T implementation, but few UNIX systems are without at least one version of Pascal, especially if the UNIX environment is one where programmers can be found. The crux of the UNIX-Pascal problem is deciding which version of Pascal is best to use. The current industry favorite is a version of Pascal that originated at Berkeley. It is an International Standards Organization (K. Jensen and N. Wirth) set, and as such is severely limited. The enhancements found in commercial Pascals in micro and mini environments are so commonplace that they will undoubtedly become part of the ISO set. Those enhancements include type string, string functions, an enlarged function library and random records.

I have seen unusual versions of Pascal (with some very strange origins) show up in UNIX. One version, called Pascal 8000, was created for the Australian Atomic Energy Commission!

The Berkeley version of Pascal, BSD Pascal, has the advantage of being offered with an interpreter as well as a compiler. The interpreter takes most of the pain out of debugging, a tremendous timesaver. It does so by offering some of the best error diagnostics I have seen from any language package.

Other features of Berkeley Pascal are a UNIX front end that allows redirection and pipes just as C and f77 do. This feature alone makes it a true UNIX language because it interfaces with the host UNIX environment rather than sitting on the sidelines like a "foreign" language. Berkeley Pascal also allows separate compilation, a feature necessary for any language used on a serious system. An interactive source code debugger is also supplied.

Debugging

The debugging of source code is so important that most UNIX systems have at least two debuggers, *dcon* and *adb*, and there are often more. The standard UNIX debuggers, which are for compiled languages only, have some problems because they deal with the assembly output of the compiler and give little if any reference to the source code. C programs output a source/assembly listing that is of some help, but the assembly reference is absolute, and the addresses you will have to deal with are relative.

Recently I was introduced to a source level debugger from a company located in Menlo Park, Calif., called CDB. CDB works directly with the source code and allows all the usual debugging tools like setting break points, single stepping, assertions, command line procedure calling, and direct access to variable names (not symbols). The debugger works with C and Berkeley Pascal, and it is being transported to other versions and languages.

Language support

Having a large number of languages on board does not make an operating system a programmer's environment. UNIX has a strong and well-deserved reputation as a programmer's world par excellence. The reputation is gained from UNIX's programming support tools. Most of these tools are aimed at the system's native assembly code and, most important of all, C. The debuggers *adb* and *dcon* deal with the native assembly as it relates to the C source code. All true UNIX-compiled languages create C code. RATFOR generates FORTRAN, which in turn creates a dialect of C. Pc, the Pascal compiler, also generates C.

Lint is a program to nitpick C code, and it does so with a vengeance. It does type checking and will pick up on such nitty-gritty things as declared, unused variables and unused returns from functions. If your program survives *lint,* it will almost certainly survive the compiler. *Cb* will beautify a C program, and *prettyp* does the same for Pascal. *Cmp* and *diff* compare equal or nearly equal source code files and report the difference. *Save* creates a backup that will not allow an older version to be written to it.

Make is a UNIX utility that literally makes the object files. It coordinates the include files, intermediate files, and separate source files and causes them to compile and link properly, regardless of which portion is modified. Unlike its human counterpart, *make* will neither duplicate effort nor omit a crucial section of the code.

One giant step beyond *make* is *sccs*, the source code control program. *sccs*, a part of *pwb*, the programmer's workbench, controls the source code to prevent older versions from sneaking in where the new version should be. It documents the changes to the code and its impact on the overall project. If more that one programmer is involved on a project *sccs* is a must.

The average UNIX system has some two score languages and language support tools. They are all interrelated and work as a true system. The overall result is a programming environment that is unmatched anywhere. Beyond program and systems development, UNIX has the ability to document the results and install online manuals for the software created as well as conventional documentation—from memos to entire books, including typesetting.

UNIX's uniqueness

The UNIX language family is unique because of the ability of its individual languages to communicate with each other and its ability to use a common function base. C programs call Shell scripts and Shell scripts call C programs with equal ease. Similarly, both the the C language and the UNIX Shell call awk, *grep*, *egrep* or *sed*. To some extent, UNIX languages almost seem as if they comprised one gigantic language, or they could be perceived as separate entities that network each other. f77 is free to use C calls or call entire C programs and return to the calling program. UNIX Pascals are also able to accomplish the same task. Some versions of C can even call FORTRAN intrinsics.

Because of the UNIX languages' ability to communicate with each other, UNIX is a unique programming environment, ideal for programmers. If you understand what is special about the team of UNIX languages, you are on your way to understanding UNIX.

Bruce Hunter is a UNIX systems administrator for Interstate Electronics in Anaheim, CA. He has authored two books— Fifty Pascal Programs *and* Understanding C*—both published by Sybex.*

CP/M-80 C Programmers . . .

Save time

. . . with the BDS C Compiler. Compile, link and execute *faster* than you ever thought possible!

If you're a C language programmer whose patience is wearing thin, who wants to spend your valuable time *programming* instead of twiddling your thumbs waiting for slow compilers, who just wants to work *fast,* then it's time you programmed with the BDS C Compiler.

BDS C is designed for CP/M-80 and provides users with quick, clean software development with emphasis on systems programming.

BDS C features include:

- Ultra-fast compilation, linkage and execution that produce directly executable 8080/Z80 CP/M command files.
- A comprehensive debugger that traces program execution and interactively displays both local and external variables by name and proper type.
- Dynamic overlays that allow for run-time segmentation of programs too large to fit into memory.
- A 120-function library written in both C and assembly language with full source code.

Plus . . .
- A thorough, easy-to-read, 181-page user's manual complete with tutorials, hints, error messages and an easy-to-use index — it's the perfect manual for the beginner *and* the seasoned professional.
- An attractive selection of sample programs, including MODEM-compatible telecommunications, CP/M system utilities, games and more.
- A nationwide BDS C User's Group ($10 membership fee — application included with package) that offers a newsletter, BDS C updates and access to public domain C utilities.

Reviewers everywhere have praised BDS C for its elegant operation and optimal use of CP/M resources. Above all, BDS C has been hailed for it's remarkable *speed.*

BYTE Magazine placed BDS C ahead of all other 8080/Z80 C compilers tested for fastest object-code execution with all available speed-up options in use. In addition, BDS C's speed of compilation was almost *twice* as fast as its closet competitor (benchmark for this test was the Sieve of Eratosthenes).

"I recommend both the language and the implementation by BDS very highly."
Tim Pugh, Jr.
in *Infoworld*

"Performance: *Excellent.*
Documentation: *Excellent.*
East of Use: *Excellent.*"
InfoWorld
Software Report Card

". . . a superior buy . . ."
Van Court Hare
in *Lifelines/The Software Magazine*

Don't waste another minute on a slow language processor. Order your BDS C Compiler today!

Complete Package (two 8" SSDD disks, 181-page manual): **$150**
Free shipping on prepaid orders inside USA.
VISA/MC, COD's, rush orders accepted.
Call for information on other disk formats.

BDS C is designed for use with CP/M-80 operating systems, version 2.2. or higher. It is not currently available for CP/M-86 or MS-DOS.

BDSoftware

BD Software, Inc.
P.O. Box 2368
Cambridge, MA 02238
(617) 576-3828

CIRCLE 3 ON READER SERVICE CARD

Bubble Sort, Insertion Sort, and Quicksort Compared

Choosing the right sorting algorithm for the right task.

By Richard G. Larson

Sorting data is an important use for computers. It is also a valuable tool in developing other applications. But selecting a sorting algorithm that will perform best for a given job can be difficult since no algorithm is appropriate in every situation. Let's look at three popular sorting algorithms—Bubble Sort, Insertion Sort, and Quicksort—and consider their relative strengths and weaknesses.

Finding data in a sorted table is an obvious case where a sorting algorithm is needed. Think how much harder it would be, for example, to find a name in the telephone book if the names were not listed alphabetically. Sorting is often used to maintain symbol tables for compilers and assemblers. (Several years ago I wrote a cross assembler which assembled Z80 code on an IBM/370. By changing the algorithm that I used to sort the symbol table, I almost doubled its speed on large assemblies.)

Another example—suppose you wanted to find all duplicates in a list of 10,000 numbers. You could do this by comparing each number with all the numbers following it in the list, but this would involve approximately 50 million comparisons. Alternatively, if you could efficiently sort the numbers into increasing order, you could find duplicates by doing the 9,999 comparisons needed to compare each number with the number immediately following it.

Bubble Sort is probably the most popular sorting algorithm among computer hobbyists. This is unfortunate. Although there is no "best" sorting algorithm, many experts believe that Bubble Sort is a leading candidate for being the worst.

Basically, Bubble Sort works by going through the array being sorted and comparing adjacent elements. If two adjacent elements are in the wrong order, they are exchanged. By passing through the array repeatedly in this manner Bubble Sort eventually gets all pairs of elements in the correct order—at which point the array is finally sorted.

An examination of what happens to individual elements shows the reason for the name Bubble Sort. On the first pass through the data, a succession of exchanges causes the largest element to "bubble up" to its final position. On the second pass, the next largest element bubbles up to its final position, and so forth.

One possible improvement in Bubble Sort involves keeping track of the location of the last out-of-order pair that was exchanged. Any data beyond this point must be in its final position and need not be examined again. In Listing 1 (printed here and also available on the *COMPUTER LANGUAGE* bulletin board service remote CP/M computer: (415) 957-9370, see disk file SORT1.LTG), the variable *LastSwap* is used to record this location. It also is used to record whether or not the data is already in order. *LastSwap* is initialized to 0 before each execution of the inner loop. If it is still 0 after the execution of the inner loop, then there were no exchanges and the data is sorted.

It can be shown that the average number of comparisons done by BubbleSort on randomly ordered data is approximately $n^2/2$, where n is the number of elements.

Bubble Sort is a relatively inefficient sorting algorithm. When sorting large arrays, a lot of sorting algorithms are many times more efficient.

For sorting small arrays, Insertion Sort is significantly faster and somewhat simpler. One of the simplest sorting algorithms available, it works by repeating the following for j running from 2 through n: (Assume $X[1]$ through $X[j\text{-}1]$ are in sorted order.) Take $X[j]$ and successively compare it with $X[j\text{-}1]$, $X[j\text{-}2]$, . . . , while moving each element larger than $X[j]$ over one position. When the first $X[i]$ is found that is not larger than $X[j]$, insert $X[j]$ immediately following it.

A standard modification (see Listing 2, also on the bulletin board service computer as disk file SORT2.LTG) is to put a dummy data element at $X[0]$ that is known to be smaller than $X[1]$ through $X[n]$. This eliminates the need for testing in the inner loop whether the loop index has reached the beginning of the array; even if it has, it will always stop when it encounters the dummy value at $X[0]$. Another minor improvement can be made by keeping the value of $X[j]$ in a separate variable rather than as an array element. We expect that a compiler will produce code to access a variable more efficiently than an array element.

The average number of comparisons done by Insertion Sort on randomly ordered data is approximately $n^2/4$. The number done in the worst case is $n^2/2$. In the best case—which is when the data is already sorted—the number of comparisons done is approximately n.

The fact that the average number of combined with the fact that the inner loop of the Insertion Sort algorithm is simpler than Bubble Sort's inner loop, suggests that Insertion Sort should be at least twice as fast as Bubble Sort. In fact, the version of Insertion Sort in Listing 2 takes less as fast as Bubble Sort. In fact, the version of Insertion Sort in Listing 2 takes less than 40% of the time taken by the version of Bubble Sort in Listing 1.

The problem with both Bubble Sort and Insertion Sort, however, is that they both move data elements to their correct positions one place at a time. It can be shown that the average distance that a data element in a randomly ordered array containing n elements must travel to its correct position is $n/3$. If n is large, and if an element moves to its correct position one place at a time, this represents a large number of operations.

Quicksort was invented by C.A.R. Hoare in the early 1960s. The basic idea is very simple but relies on a concept called recursion: the ability of a procedure or subroutine to call itself. Hoare remarked on the importance of recursion in a lecture he gave the day he received the 1980 ACM Turing award, "I first learned about recursive procedures [around Easter 1961] and saw how to program the sorting method which I had earlier found such difficulty in explaining."

Quicksort sorts segments of an array. To sort an entire array, it sorts the segment from 1 through n. It works by taking a partitioning element from the array and rearranging the array so that all elements on the left-hand side preceed the partitioning element, all the elements on the right-hand side follow the partitioning element, and the partitioning element is placed between the left- and right-hand sides. It then recursively sorts the left- and right-hand segments.

Quicksort's advantage over Bubble Sort and Insertion Sort comes from the fact that the partitioning method exchanges non-adjacent elements, causing each element to migrate to its correct position in the array with many fewer operations.

The key problem is identifying a good partitioning element. Ideally, about half of the elements in the segment being sorted should preceed it and about half should follow it. A simple implementation of Quicksort is given in Listing 3 (printed here and also available as disk file SORT3.LTG on the buletin board service). This implementation is a pedagogical one and is not intended to be used in an application program. It consists of a recursive procedure, *BQS*, which does the actual sorting, followed by a procedure, *BasicQuickSort*, which sets things up and calls *BQS*. The implementation in Listing 3 simply uses the first elements of the segment as a partitioning element. Assuming the array elements are in random order, this is not a bad choice. However, if the array is not in random order (e.g., it already happens to be sorted), this choice can be disastrous.

Once the partitioning element is identified, the segment is scanned from the left until the first element not preceeding the partitioning element is encountered. The segment is then scanned from the right, stopping at the last element not following the partitioning element. When these two elements are located, they are exchanged. The two scanning processes continue until they meet in the middle of the segment. Partitioning is completed when the partitioning element is inserted at the meeting point.

This version of Quicksort can be shown to take about $2n \log_2 n$ comparisons on randomly ordered data. The fact that the function $\log_2 n$ is much smaller than n for large values of n (e.g., $\log_2 1000$ is about 10) implies that, for large n, Quicksort will do many fewer comparisons on the average than Bubble Sort or Insertion Sort. In the worst case, Quicksort does about $n^2/2$ comparisons, which is as bad as Bubble Sort.

What is worrisome about this version of Quicksort, however, is that the algorithm is at its worst processing already sorted data. When the array is in order, choosing the first element of a segment of a size s as a partition element gives two sub-segments of a size 0 and size s-1. Quicksort operates most efficiently when partitioning results in two sub-segments of nearly equal size. The version of Quicksort in Listing 3—which is the most easily understood version—should not be used unless you are certain that the array to be sorted is in random order.

An enhanced version of Quicksort is given in Listing 4 (available only on the bulletin board service as file SORT4.LTG). Again, I present a recursive routine, *QS*, followed by a routine, *Quicksort*, which calls *QS* and then does some final computations. Table 1 shows that these improvements give small decreases in run time over the version presented in Listing 3. The enhanced version's most important advantage—which does not appear in the table—is that it is less likely to behave badly on non-random data.

The most important feature of a Quicksort implementation is the selection of the partitioning element. Rather than simply taking the first element of the segment—on the assumption that it is randomly located within the partition—take the median element of the first, middle, and last elements of the segment. If the array is already ordered (or nearly ordered), this is obviously a good choice. For a randomly ordered array, it is also a good choice and gives a partitioning element that is closer to the middle point than the first element. (The average number of comparisons goes down from $2n \log_2 n$ to $(12/7)n \log_2 n$.) Some people like to partition the segment using the numerical average of the first and last elements, but this won't work very easily if you are sorting non-numeric data.

Another improvement can be made by noting the fact that the procedure *BQS* ends with a recursive call. This call can be removed by setting the values of the parameters appropriately and re-executing the body of the procedure. This is often described as tail recursion. It is also desirable to limit the depth of recursive calls because deep recursive calls use more memory and a recursive call is often more time consuming than re-executing the procedure block. In the procedure *QS*, this is achieved by doing the recursive call on the shorter segment and using tail recursion on the longer segment. Doing this guarantees the depth of recursion will never exceed $\log_2 n$.

The final improvement comes from the fact that for very small segments, Insertion Sort is more efficient than Quicksort because of Quicksort's complexity. This means that when the array is almost sorted, it is better to abandon Quicksort and finish the job with Insertion Sort. By doing this you can sort data elements most efficiently since every element is near its final position. This is done in the procedure *QS*; *QS* does not sort a segment unless its length is greater than M—a constant in the procedure. The value of M depends on the specific implementation. In Donald E. Knuth's *The Art of Computer Programming, vol 3, Sorting and Searching* (pp. 119-122), you can see how the best value of M can be found in a sample assembly language implementation. Knuth uses information on instruction timing and some subtle mathematical analysis of the probability of taking different paths in the procedure.

A simpler approach is to find the value of M by experimentation. For example, with my machine and my compiler running this procedure, the correct value of M was 12. However, the value of M is not critical. The execution times for this version of *QS*, sorting an array of 5,000 elements, with $M = 6$ and $M = 18$, were only about 2% greater than with M equal to 12. M equals 10 is usually a safe choice.

So, which sorting algorithm is the best to use? If the array to be sorted is small (e.g., substantially less than 100 elements), or if the data to be sorted is already nearly ordered, then Insertion Sort is a reasonable choice. Otherwise, use an enhanced version of Quicksort like the one given in Listing 4. Conventional wisdom says this is the fastest "average case" sorting algorithm and the choice of the median element as the partitioning element gives good protection against cases where the data to be sorted is in extremely random order.

Mathematical proofs of many of the assertions made in this article can be found in Donald Knuth's book *The Art of Computer Programming, vol 3, Sorting and Searching*, published by Addison-Wesley, 1975. Anyone who is interested in the more subtle problems connected with sorting algorithms will find an incredible wealth of information in that volume.

The algorithms presented in Listing 1, 2, 3, and 4 were compiled using Microsoft Pascal (version 3.13) and run 10 times each on various sizes of arrays filled with random real numbers on an IBM XT with an 8087 chip under DOS 2.0. The results are presented in Table 1. Since the resolution of the system clock under DOS is only 0.05 sec, the timing for the two versions of Quicksort for the smallest values of n should not be taken very seriously.

```
procedure BubbleSort (var X : DataArray; n : integer);
var
   j,
   Limit,   { data at position above here is in final position}
   LastSwap{ holds position of last data pair swapped }
      : integer;
begin
   Limit := n;
   while not (Limit = 0) do
      begin
         LastSwap := 0;
         for j := 1 to Limit-1 do
            if X[j] > X[j+1] then
               begin
                  swap(X[j], X[j+1]);
                  LastSwap := j
               end;
         Limit := LastSwap
      end
end;
```

Listing 1.

```
procedure InsertionSort (var X : DataArray; n : integer);
var
   j, i : integer;
   Z {temporarily holds X[j] while X[j-1],...are being moved up}
      : real;
begin
   X[0] := SmallerThanAnything;
   for j := 2 to n do
      begin
         Z := X[j];
         i := j - 1;
         while (Z < X[i]) do
            begin
               X[i+1] := X[i];
               i := i - 1
            end;
         X[i+1] := Z
      end
end;
```

Listing 2.

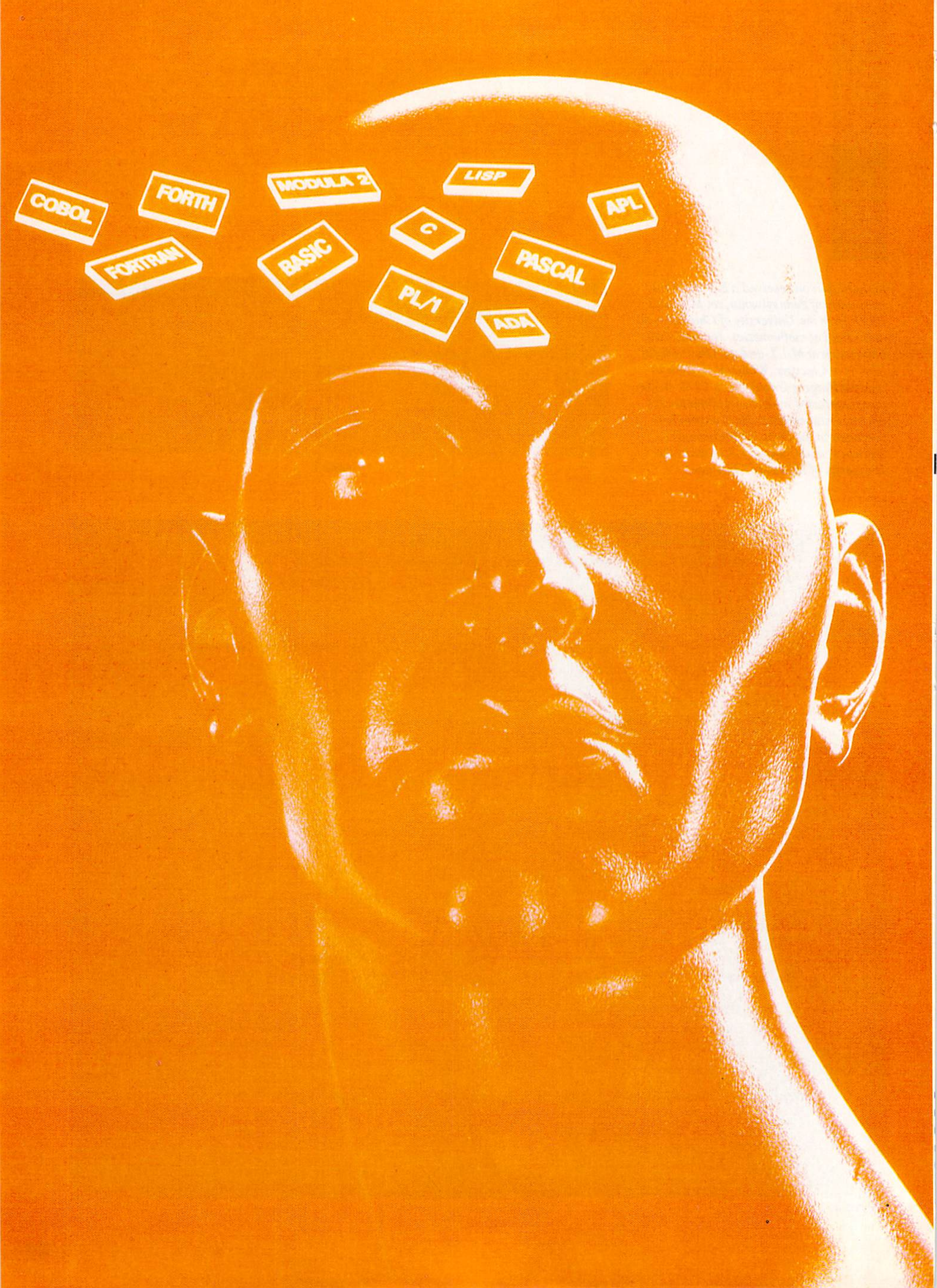
Richard Larson received a B.S. from the University of Pennsylvania, an M.S. and Ph.D. from the University of Chicago—all in the field of mathematics. He then went on to teach at M.I.T. and the University of Illinois at Chicago. While on sabbatical leave at Rutgers University in 1974, he became interested in applying computers to abstract mathematics. He is currently a professor in the Dept. of Mathematics, Statistics, and Computer Science at the University of Illinois at Chicago.

n	Bubble	Insertion	Simple Quick	Enhanced Quick
125	1.64	0.63	0.16	0.16
250	6.38	2.35	0.37	0.35
500	25.48	9.24	0.81	0.76
1000	102.40	36.50	1.80	1.69
2000	411.57	145.99	4.02	3.67

Table 1.

```
procedure BQS (var X : DataArray; i, j : integer);
{ procedure called by BasicQuickSort to do actual sorting }
var
    left, right : integer;
    Z : real;
begin
{Partition the array segment X[i]..X[j] using X[i] as partitioning element}
    Z := X[i];
    left := i;
    right := j + 1;
    while (left < right) do
        begin
            repeat
                left := left + 1
            until X[left] >= Z;
            repeat
                right := right - 1
            until X[right] <= Z;
            if left < right then
                swap(X[left], X[right])
        end;
    X[i] := X[right];
    X[right] := Z;
{At this point we have: for all k < right, X[k] <= X[right];
                        for all k > right, X[k] >= X[right].
 Recursively sort the segments X[i]..X[right-1] and X[right+1]..X[j].}
    if i < right-1 then
        BQS(X, i, right-1);
    if right+1 < j then
        BQS(X, right+1, j)
end;

procedure BasicQuickSort (var X : DataArray; n : integer);
begin
    X[n+1] := LargerThanAnything;
    BQS(X, 1, n)
end;
```

Listing 3.


COBOL
FORTH
MODULA 2
LISP
APL
C
FORTRAN
BASIC
PASCAL
PL/1
ADA

Improve Your Programming with Structured Techniques

By Dr. Joseph B. Rothstein

So much as been written and said on the subject of structured programming that it's important to consider exactly what this concept means.

To some people, structured programming has become almost a religion, a virtual salvation of the programmer's soul. To others, it's a conspiracy intended to snuff out programmers' creativity and turn them into code-spewing drones.

Both of these views are, of course, equally ridiculous. The concepts and techniques of structured programming—first suggested by Edsger Djikstra, then refined by Niklaus Wirth, Donald Knuth and others—are an attempt to define an orderly approach to designing, implementing, debugging and testing computer programs. From a disciplined and well thought out methodology comes programs that are well organized, reliable and maintainable.

Far too many programmers (especially student programmers) think only about writing code when they undertake a programming project. In fact, studies have shown that writing code forms only a small part of the program development process. Actually, much of a program designer's time is spent determining exactly what a program should do and how it should do it. And after the code is written, a programmer can easily become trapped in a seemingly endless cycle of testing and debugging that can take far longer than the time spent coding.

With the advent of ever cheaper hardware, the costs associated with programming have assumed a greater proportion of EDP department budgets. The bulk of these programming expenses goes not for new programs but for maintaining, modifying, and extending existing programs. Most of these costs, in turn, arise from the unfortunate responsibility of having to wade through a tangle of existing code to unravel a program's fundamental logic.

Structured programming seeks to remedy those failures of the "program as you go" approach. By pursuing an orderly, methodical orientation that is the same for each application or task, the programmer can minimize the tediousness and drudgery associated with writing code and focus more on the real problems and their solutions.

Rather than stifling creativity, structured programming can minimize the burdens that result from disorganization and allow the programmer time to dwell on the more satisfying aspects of programming—the design of elegant algorithms and expeditious data structures.

Structured programming, then, is an orderly approach to the program development cycle and an associated set of steps and techniques that implement this viewpoint. Diligently applied, it can minimize the drudgery usually associated with programming and emphasize instead those activities in which creativity, style, and insight can have the most positive impact.

To illustrate the concepts involved in structured program design, let's use a sample application and follow its development from the initial stages to the completed code. Before beginning, however, it is important to emphasize a couple of points. First, the use of structured programming is not a panacea and does not guarantee good programs. It is possible to write code poorly regardless of its structured nature. When all is said and done, there is no substitute for intelligence, talent, and the education of experience.

Second, and probably more important, is that the techniques of structured programming are language independent. Certainly, the design of some programming languages is better suited to encouraging the use of structured techniques than others. Pascal generally comes to mind in this regard; after all, one of Kathleen Jensen's and Niklaus Wirth's goals was to implement these concepts in their design of Pascal. But by exercising some planning and thought, the programmer can design and implement structured programs in any language—including FORTRAN, assembler, or that bane of programming purists, BASIC.

Structured programming may be considered a four-step process involving three techniques. The four steps may be summarized as: goal, specifications, pseudocode, and program code. The techniques involved are top-down design, structural decomposition, and modularity. We will deal with each in turn.

Decide on the objective

Regardless of the programming language, the first step must be to determine the goal of the program. This simple yet crucial step is often too easily dismissed, which can have catastrophic results. Needless programs are written, tempers become frayed, and lawyers obtain business when programmers write code without determining the objective involved.

Careful consideration of the objective of a given program will lead to an easier, more straightforward solution. For example, the response to a request for "a program that will straighten out my accounting" may not be a program at all but rather a better accounting procedure.

While the goal of a program may be stated in general terms, it must clearly indicate the nature and scope of the project. If a person said, for example, that he or she wanted to write "a sales report program," this would not be an adequate description. But "a report, run weekly, which shows sales for each hour of each day of the preceding workweek and generates cumulative totals by hour and day, based on hourly sales data that is entered by the user and validated by the program" describes the goal clearly and succinctly.

At this stage, it is not important to know the wording of the report's title or the nature of the data validation methods. It is enough to realize that data entry and validation will form a significant part of the program and that the purpose of the report is to show hourly and daily sales figures for a week long period.

Starting with the big picture, rather than being concerned with minor details, is the central principle of top-down designing. In essence, this approach stresses working progressively from the general level down into the details rather than beginning with detailed descriptions of prompts, file layouts, and other low-level concerns. Failure to see the forest for the trees can be a problem in any activity, and particularly in programming.

Consider the specifications

Once the goal is clearly established, the programmer may begin to consider the specifications for the program. Just as the goal statement is language independent, so should the specifications need no reference to a particular language.

Specifications may take a variety of forms, but one of the most useful is the Input-Process-Output model of a program. In this view, a program may be considered as a "black box" (process) that uses available data (input) to generate some worthwhile result (output).

Make a list of the available data and describe the desired output as fully as possible. Later on, the input data list will form the core of documentation—such as sample input screens and a data dictionary—while the output data list can lead to file layouts and sample report forms.

In our sample sales report, we'll certainly need the hourly sales figures for each day. We might also need the weekday name or date on which we begin. There are many data entry alternatives, and it's important to clarify not only what input is necessary but also what is unnecessary. If we assume, for instance, that each report will cover Monday through Friday (inclusive) and the hours of 9 a.m. until 5 p.m., we can eliminate a considerable amount of data entry—always a desirable ideal. On the other hand, we may know only the starting date, so we might need to determine what day of the week it is and generate the names of the subsequent days within the program.

Our approach does not require that the input specifications be complete in every detail before undertaking the description of the process by which input will be transformed into output. In the early stages that description must remain flexible. We may find that additional input is required or that some input is unnecessary and may be eliminated, and then we would adapt the specifications accordingly. It is far easier to change a program specification than to debug finished code. By stressing an investment of time spent in the design phase, structured programming can pay big dividends later during debugging.

We can generate a process specification by starting with our goal statement and successively refining it by using a technique called stepwise (or structural) decomposition. This approach suggests that any large or complex task can be progressively refined (decomposed) into a series of simpler tasks. It also helps identify the major design issues involved so that they may be addressed immediately or deferred until their details become clear, while still accommodating them in the overall design plan.

Keep in mind that our overview of the program is already decomposed into three stages:

1) Get input from user
2) Perform necessary processing
3) Print results

We can now focus on each of these in turn. Stage 1 might be further decomposed into the following series of activities:

A) Print an introductory message for the user
B) Prompt the user for one hour's sales data
C) Validate that the value entered is acceptable
D) Store the value
E) Repeat for each hour in the reporting period

Clearly, step C is vaguely defined at this point. But that should not be of concern because we know the tasks to be performed, though we don't yet know the details of how each task will be performed. The important consideration at this level of design is that we be able to identify the major tasks, the order in which they are to be performed, and the control structures that govern them. Those control structures should also be described in general terms, as above, rather than trying to force them to conform to the syntax for control structures in a particular programming language.

This overview of tasks, once completed for all three portions of our initial design, serves two complementary purposes. In addition to serving as our process specification, it will subsequently become the "driver" of our program.

We can view a program as composed of two portions: the driver, which implements the high-level design, and subprograms invoked by the driver, which implement the details of that design. In FORTRAN or BASIC, for example, those subprograms would be subroutines that are invoked by the driver with a *GOSUB* statement. Pascal uses procedures and C uses functions, but both serve the same purposes: to physically separate the high-level design from the implementation details of that design, and to localize each task to a single subprogram.

Ideally, the driver should contain only three classes of statements: comments, control structures, and calls to subprograms. Sometimes, of course, it is necessary to set up parameters in the driver for passage to subprograms, but with careful planning the use of even these statements can be minimized.

The concept behind single-task subprograms is part of the third technique of structured programming. Modularity suggests that any program can and should be implemented as a series of free-standing, individually-testable subprogram modules, each of which performs a single task. Free-standing refers back to the black box concept of programs—that if a program module is given its required input it will generate its output in a predictable fashion. If this is the case, each module can be tested (independent of the rest of the program) by force feeding it test input data for which the output should be known and then making sure that the module generates the expected output.

The modular approach has numerous advantages. It helps us isolate the locus of any bugs or anomalies, find the appropriate routines when we subsequently need to make modifications or extensions to the program, implement our program incrementally, and test our work as we progress rather than by trying to test and debug the entire program at once.

For any non-trivial program, it is only possible to prove the existence of bugs, not their absence. It follows, then, that by reducing a complex program to a series of trivial modules and testing each module, we can be more confident that we have done a thorough debugging job.

Write the pseudocode

We can now begin the third step in our structured methodology— generating pseudocode for the driver and each of the subprograms specified. Pseudocode means "false code" or "almost code" in the sense that it should look similar to a finished program.

In writing pseudocode, however, we are not bound by the syntactical constraints of any particular programming language. Therefore, we can continue to develop our program in a language-independent fashion and later transform our pseudocode into the programming language of our choice.

The pseudocode for the driver portion of our program might appear as presented in Figure 1. This pseudocode should suggest the overall structure of the tasks in our program, the order in which they will be performed, and the control structures required. Careful attention to the control structures can help ensure an orderly, smooth transition from one subprogram to the next. It may also serve to minimize or eliminate the use of unconditional *GOTO* statements—considered a grave offense by structured programming purists.

```
Print instructions to user.
For each workday
   print a prompt using the name of that workday.
   For each hour of the day
     Repeat until a valid amount is entered:
       Print a prompt indicating the hourly period.
       Print a prompt for the sales amount.
       Get a sales amount entry from the user.
       Validate amount entered.
       If invalid, print an error message.
     Store the value in the sales table.
Perform the calculations.
Print the report.
```

Figure 1.

```
Using the sales amount entered by the user,
  Set a flag if:
    the amount is less than zero;
    the amount exceeds some predetermined maximum; or
    the integer value of the decimal portion exceeds 99.
```

Figure 2.

Super assemblers plus the world's largest selection of cross assemblers!

Z-80
Macroassembler $49.50

Power for larger programs! This 2500AD macroassembler includes:

- Zilog Z-80 Macroassembler (with the same powerful features as all our assemblers)
- powerful linker that will link up to 128 files
- Intel 8080 to Zilog Z-80 Source Code Converter (to convert all your Intel source to Zilog Syntax in one simple step)
- COM to Hex Converter (to convert your object files to Hex for PROM creation, etc.)
- 52 pages User Manual

8086/88 Assembler
with Translator $99.50

Available for MSDOS, PCDOS, or CPM/86! This fully relocatable macroassembler will asemble and link code for MSDOS (PCDOS) AND CPM/86 on either a CPM/86 or MSDOS machine. This package also includes:

- An 8080 to 8086 source code translator (no limit on program size to translate)
- A Z-80 to 8086 translator
- 64 page user manual
- 4 linkers included:
  - –MSDOS produces .EXE file
  - –CPM/86 produces .CMD file
  - –Pure object code generation
  - –Object code and address information only

Linker features:

- Links up to 128 files
- Submit mode invocation
- Code, Data Stack and extra segments
- Handles complex overlays
- Written in assembly language for fast assemblies.
- MICROSOFT .REL format option

Z-8000 Cross Development Package $199.50

Instant Z-8000 Software! This package allows development and conversion of software for the Z8001, 8002, 8003 and 8004 based machines on a Z-80, Z-8000 or 8086 machine. This powerful package includes:

- a Z-80/8080 to Z-8000 Assembly Language Source Code Translator
- Z-8000 Macro Cross Assembler and Linker

The Translators provide Z-8000 source code from Intel 8080 or Zilog Z-80 source code. The Z-8000 source code used by these packages are the unique 2500AD syntax using Zilog mnemonics, designed to make the transition from Z-80 code writing to Z-8000 easy.

All 2500 AD Assemblers and Cross Assemblers support the following features:

Relocatable Code — the packages include a versatile Linker that will link up to 128 files together, or just be used for external reference resolution. Supports separate Code and Data space. The Linker allows Submit Mode or Command Invocation.

Large File Handling Capacity —the Assembler will process files as large as the disk storage device. All buffers including the symbol table buffer overflow to disk.

Powerful Macro Section— handles string comparisons during parameter substitutions. Recursion and nesting limited only by the amount of disk storage available.

Conditional Assembly—allows up to 248 levels of nesting.

Assembly Time Calculator— will perform calculations with up to 16 pending operands, using 16 or 32 Bit arithmetic (32 Bit only for 16 Bit products). The algebraic hierarchy may be changed through the use of parentheses.

Include files supported—

Listing Control—allows listing of sections on the program with convenient assembly error detection overrides, along with assembly run time commands that may be used to dynamically change the listing mode during assembly.

Hex File Converter, included —for those who have special requirements, and need to generate object code in this format.

Cross reference table generated—

Plain English Error Messages—

System requirements for all programs: Z-80 CP/M 2.2 System with 54k TPA and at least a 96 column printer is recommended. Or 8086/88 256k CP/M-86 or MSDOS (PCDOS).

Cross Assembler Special Features

Z-8—512 User defined registers names, standard Zilog *and* Z-80 style syntax support.

8748—standard Intel *and* Z-80 style syntax supported.

8051—512 User defined register or addressable bit names.

6800 Family—absolute or relocatable modes, all addressing modes supported, Motorola syntax compatible.

6502—Standard syntax or Z-80 type syntax supported, all addressing modes supported.

8086 and Z-8000 XASM includes Source Code Translators

	Z-80 CP/M®	ZILOG SYSTEM 8000 UNIX	IBM P.C. 8086/88 MSDOS	IBM P.C. 8086/88 CP/M 86	OLIVETTI M-20 PCOS
8086/88 ASM			$ 99.50	$ 99.50	
8086/88 XASM	$199.50	$750.00			$199.50
16000(all) XASM *new*	199.50	750.00	199.50	199.50	199.50
68000 XASM *new*	199.50	750.00	199.50	199.50	199.50
Z-8000™ ASM		750.00			299.50
Z-8000 XASM	199.50		199.50	199.50	
Z-80 ASM	49.50				
Z-80 XASM		500.00	99.50	99.50	99.50
Z-8 XASM	99.50	500.00	99.50	99.50	99.50
6301(CMOS) *new*	99.50	500.00	99.50	99.50	99.50
6500 XASM	99.50	500.00	99.50	99.50	99.50
6502 XASM	99.50	500.00	99.50	99.50	99.50
65CO2(CMOS) XASM *new*	99.50	500.00	99.50	99.50	99.50
6800,2,8 XASM	99.50	500.00	99.50	99.50	99.50
6801,03 XASM	99.50	500.00	99.50	99.50	99.50
6805 XASM	99.50	500.00	99.50	99.50	99.50
6809 XASM	99.50	500.00	99.50	99.50	99.50
8748 XASM	99.50	500.00	99.50	99.50	99.50
8051 XASM	99.50	500.00	99.50	99.50	99.50
8080 XASM	99.50	500.00	99.50	99.50	99.50
8085 XASM *new*	99.50	500.00	99.50	99.50	99.50
1802 XASM *new*	99.50	500.00	99.50	99.50	99.50
F8/3870 XASM *new*	99.50	500.00	99.50	99.50	99.50
COPS400 XASM *new*	99.50	500.00	99.50	99.50	99.50
NEC7500 XASM *new*	99.50	500.00	99.50	99.50	99.50
NSC800 *new*	99.50	500.00	99.50	99.50	99.50
Subtotal	$ ________	$ ________	$ ________	$ ________	$ ________

Name ____________________
Company ____________________
Address ____________________
City ____________ State ________ Zip ________
Phone ____________________ Ext. ________
Make and model of computer system ____________________
☐ C.O.D. *(2500AD pays C.O.D. charges)*
☐ VISA or MasterCard #, Exp. Date (mo./yr.)
____

TO ORDER. **Simply circle the product or products you want in the price columns above, enter the subtotal at the bottom of that column and add up your total order. Don't forget shipping/handling.**

Total $ ________

Check one:
☐ 8″ Single Density
☐ 5¼″ Osborne
☐ IBM P.C.
☐ Cartridge Tape
☐ Apple (Softcard)

shipping/handling ($6.50 per unit, $20.00 per unit for Int'l. airmail) $ ________

Total Order $ ________

CP M is a registered trademark of Digital Research, Inc.

Signature ____________________

2500AD SOFTWARE INC.

P.O. Box 441410, Aurora, CO 80014, 303-752-4382 TELEX 752659/AD

Now we can focus on the pseudocode for each task in turn. As an example, let's look at the validation routine. It might appear as the pseudocode routine in Figure 2. This pseudocode suggests that the driver must pass the amount entered to the validation routine as a parameter, and that the validation routine must indicate the presence (and perhaps the nature) of any errors. We will implement the routine as a Pascal function, but it could just as easily be coded in BASIC, FORTRAN, or even assembly language.

The final step—program code

Given similar pseudocode and parameter information for each of the routines, we can then perform the final step in the implementation of our program—translating first the driver and then the routines themselves into Pascal code. The resulting pro-

```
BEGIN     (* main *)

  initialize(salestable,hourlytotal,dailytotal);
  intro;  (* print instructions for user *)

  FOR dayindex := (* user-defined data type *) mon TO fri DO
    BEGIN

      dayprompt(dayindex);  (* print day name for user *)
      FOR hourindex := (* user-defined *) nine TO four
        BEGIN

          REPEAT
            hourprompt(hourindex); (* print hour prompt for user *)
            getinput(hoursales);
            IF NOT valid(hoursales) THEN
              printerror
          UNTIL valid(hoursales);

          store(salestable,hoursales)
        END            (* single hour sales input loop *)

    END;               (* single day loop *)

calculate(salestable,hourlytotal,dailytotal);
report(salestable,hourlytotal,dailytotal)

END.                     (* program *)

     The Pascal code for our validation function might be:

FUNCTION valid(sales:REAL) : BOOLEAN;
  BEGIN
    IF (sales > maxsale) OR (sales < 0) THEN
      valid := FALSE;
    ELSE
      valid := TRUE;
  END;    (* function valid *)
```

Listing 1.

gram is seen in Listing 1. (Reserved words are in upper case, user-defined names are in lower case, and comments are enclosed within the "(* . . . *)" symbols.)

By writing such trivial functions, procedures, or subroutines, we can keep our code as simple as possible despite the overall complexity of our program, that might consist of hundreds of such fragments. In addition, if we thoroughly document our subprograms and save them in libraries, we may need only to look in our "programmer's toolkit" for debugged and tested solutions to problems which might arise in the future, thus saving ourselves from having to re-invent the wheel with each new program.

In the space of such a short article it is difficult to do justice to a topic as straightforward yet far-reaching as structured programming. Each programmer develops a unique style, and I have probably adapted structured programming to my own style as much as the other way around.

I believe that the concepts involved are not rules but guidelines. After observing the work of scores of professionals and hundreds of programming students using a variety of languages, I am convinced that using the guidelines as a point of departure seems to result in better programs by any standard of judgement. In a programmer's continuing education, there is no substitute for writing and studying as many programs as possible. But structured programming can accelerate the learning process by providing a consistent, unifying framework for understanding the process of programming and the working of programs. ■

Dr. Joseph B. Rothstein is president of Hanahoa Software Corp., specializing in microcomputer applications for the business environment. He has lectured, taught, and published extensively on subjects relating to programming languages, applications, and social impacts of computing. He earned his Ph.D. from the State University of New York at Buffalo, specializing in computer applications to the musical arts.

Still Fixing Bugs The Hard Way?

LTC
Sergio Roffo

Ready to take the sting out of debugging? You can with Pfix86™ and Pfix86 Plus™, the most advanced dynamic and symbolic debuggers on the market today for PC DOS and MS-DOS™ programmers.

What other debugger offers you an adjustable multiple-window display so you can view program code and data, breakpoint settings, current machine register and stack contents all at the same time? And, an in-line assembler so you can make program corrections directly in assembly language. Plus, powerful breakpoint features that allow you to run a program at full speed until a loop has been performed 100 times, or have the program automatically jump to a temporary patch area.

Or maybe you're tired of searching through endless piles of listings for errors? With Pfix86 Plus you won't have to. You can locate instruction and data by the symbolic name and using the symbolic address. Handle larger, overlayed programs with ease. And, Pfix86 Plus is designed to work with our Plink86™ linkage editor.

But that's not all. With a single keystroke you can trace an instruction and the action will be immediately reflected in code, data, stack, and register windows. Pressing a different key will elicit a special trace mode that executes call and loop instructions at full speed, as though only a single instruction were being executed.

And you get an easily accessible menu that makes the power of our debuggers instantly available to the new user, but won't inhibit the practiced user.

So, why struggle with bugs? Pfix86 by Phoenix. Pfix86 $195. Pfix86 Plus $395.
Call (1) 800-344-7200, or write.

Phoenix

Phoenix Software Associates Ltd.
1420 Providence Highway Suite 260
Norwood, MA 02062
In Massachusetts (617) 769-7020

Pfix86, Pfix86 Plus and Plink 86 are trademarks of Phoenix Software Associates Ltd.
MS-DOS is a trademark of Microsoft Corporation

CIRCLE 38 ON READER SERVICE CARD

COBOL
Pride and Prejudice

By Robert Wagner

> *"I am a jealous God, visiting the iniquity of the fathers upon the children unto the third and fourth generation."*
>
> *—Exodus 20:5*

Do you think COBOL is a dinosaur? Most micro people do. Why? I suspect the reason has to do with the psychology driving the micro movement, not with the merits of the language.

For 20 years, only the "big" people—corporations and government agencies—could afford a computer. It became a symbol of power and impersonality. It was kept in a special shrine where only authorized people could enter, and there was absolutely no eating or smoking in its presence.

Suddenly all that changed. Computers were cheap. Little people could afford their own private deus ex machina. They bought computers by the millions not because they really needed them (who really needs a home computer?) but because of symbolism. They were buying power and control over their own destinies. They thought they were buying an expert system that could solve problems they personally couldn't.

Like an adolescent tasting freedom for the first time, they overreacted. If something looked or smelled like the old way, the parents' way, it must be bad. Micros weren't just small computers, they were a revolution. They were going to make mainframes into dinosaurs. And since the old timers spoke COBOL, that too went into the scrap heap.

Revolutions do not succeed just because they are technically feasible. Failure of the old system is also required. For example, television was technically feasible in the 1920s but didn't replace movies until the 1950s. Why? Because movies still "worked." Air travel was expensive until the railroad became senile in the mid 1960s. Electronic banking hasn't replaced the checkbook because paper shuffling still works. Micros and Pascal will be around, but they will not replace mainframes and COBOL until these become senile.

An estimated $50 billion worth of production COBOL is running on mainframes. Maybe it's not worth that much, but that's what it cost. It's more than all the other languages put together. About 60-70% of the new application code is being written in COBOL (future job hunters take note).

Why do companies prefer COBOL? Tradition? It's the only language they know? If you ask them, they say that it's the most "maintainable" language. Other languages offer speed of development at the sacrifice of clarity. A program that cannot be understood and changed by someone else (possibly someone mediocre) is an expense, not an asset. Listing 1 shows the same logic written in COBOL, C and Forth. Which do you find easiest to read? I submit that anyone can understand COBOL. Can anyone understand Forth intuitively? I can't. Do braces and semicolons make C easier to read? For a compiler yes—for a human no.

I have another theory. At least it's the reason I like COBOL. If you're not familiar with the bicameral brain concept, it says that you have dual processors inside your head. The left brain deals with things, the right brain with people. The left brain solves problems from the bottom up, the right brain from the top down. The left brain is a technician, the right brain a designer ("design engineer" is an oxymoron). The left brain wants evolution, the right brain revolution. In most people, one or the other is dominant. In a minority, they're dual.

My theory is that really good programmers are "balanced," while the hordes of average programmers are left-brain dominant. (A right-brain person can't make it in the computer world—he or she usually becomes a manager.) The left brain is a bad designer. For examples, look at the specs for OBJ files, the 8087 interface, SNA, or most system software. The tendency is to take a simple problem and make it complicated.

The right brain is a bad technician. It wants simple answers to complex problems. For example, just talk with any salesperson. The key to success is ambivalence. The word commonly means indecisiveness, but to me it means fast (several times per second), cooperative, and having a transparent alternation between left and right.

A computer, it is said, is a machine that's not as intelligent as a human being but more intelligent than a programmer. But "programmer" embraces many personalities. That's why we have so many languages. There's one for each style. Left-brained programmers favor languages that are mechanistic, technical and unnecessarily formal. A balanced programmer wants a language that's technical enough to do anything (most fourth generation languages can't) but informal enough that the right brain can understand it too. I believe COBOL comes closest to meeting this requirement. It's also my personal order of preference.

Today the majority of micro users are "little" people: small businesses, researchers, enthusiasts (hackers). They see the micro as a complete data processing system, a little mainframe.

But in the next two to three years, "big" people will become an important factor. They will see the micro as a component of a larger system—specifically a user interface, a smart terminal. When the screen is only a move instruction away and the CPU is idle 99% of the time, it's perfect for building screens. Let the mainframe, with its 3-meg channels, handle I/O, and let the micro handle the user. Now the only problem is how to transport CICS COBOL.

Four COBOL compilers are available for the IBM PC. Table 1 gives their approximate cost and speed of generated code compared with the other languages. Note that they are all expensive. One—mbp—is notably faster than the other three. That's because it generates real machine language while the other three produce p-code. BOS is COBOL-like but very non-standard.

Most Program Editors Are Shockingly Primitive.

Sergio Raffo

Use Pmate™ once, and you'll never go back to an ordinary text editor again. Pmate is more than a powerful programmer's text processor. It's an interpretive language especially designed for customizing text processing and editing.

Just like other powerful editors, Pmate* features full-screen single-key editing, automatic disk buffering, ten auxiliary buffers, horizontal and vertical scrolling, plus a "garbage stack" buffer for retrieval of deleted strings. But, that's just for openers.

What really separates Pmate from the rest is macro magic. A built-in macro language with over 120 commands and single-keystroke "Instant Commands" to handle multiple command sequences. So powerful, you can "customize" keyboard and command structure to match your exact needs.

Get automatic comments on code. Delete comments. Check syntax. Translate code from one language to another. Set up menus. Help screens. You name it.

And, Pmate has its own set of variables, if-then statements, iterative loops, numeric calculations, a hex to decimal and decimal to hex mode, binary conversion, and a trace mode. You can even build your own application program right inside your text processor.

So, why work with primitive tools any longer than you have to? Pmate by Phoenix. $225. Call 1-800-344-7200. Or Write.

Phoenix

Phoenix Software Associates Ltd.
1420 Providence Highway Suite 260
Norwood, MA 02062
In Massachusetts (617) 769-7020

*Pmate is designed for microcomputers using the Intel 8086 family of processors, and running MS-DOS™ A custom version is available for the IBM PC, TI Professional, Wang Professional, DEC Rainbow, and Z80 running under CP/M™

Pmate is a trademark of Phoenix Software Associates Ltd.
MS-DOS is a trademark of Microsoft Corporation. CP/M is a trademark of Digital Research, Inc.

CIRCLE 39 ON READER SERVICE CARD

```
               (a)
            COBOL program

     PROGRAM-DRIVER.
          PERFORM ONE-ITERATION 10 TIMES.
          DISPLAY PRIME COUNT 'PRIMES'.
          STOP RUN.
     ONE-ITERATION.
          MOVE ZEROS TO PRIME-COUNT, FLAGS.
          PERFORM ONE-TRIAL
              VARYING N FROM 1 BY 1
              UNTIL N GREATER THAN 8191.
     ONE-TRIAL.
          IF FLAG (N) EQUAL TO ZERO
             COMPUTE PRIME = N + N + 1
             ADD 1 TO PRIME COUNT
             DISPLAY 'FOUND PRIME' PRIME
             COMPUTE X = N + PRIME
             PERFORM MARK-OUT
                 VARYING X FROM X BY PRIME
                 UNTIL X GREATER THAN 8191.
     MARK-OUT.
         MOVE '1' TO FLAG (X).
```

```
               (b)
            C program

main ()  {
for (iter = 1; iter <=10; iter++)  {
     count=0;
     for(i = 0; i<= size; i++)
          flags[i] = true;
     for(i = 0; i<= size; i++)  {
          if(flags[i])  {
               prime = i + i + 3
/*             printf("\n%d",prime);/*
               for (k=i+prime; k<=size; k+=prime)
                    flags[k] = false
               count++;
          }
     }
}
     printf("\n%d" primes.",count);
}
```

```
               (c)
            Forth program

     0 ( 0 COUNT ) SIZE 0
       DO  FLAGS I + C@
         IF I DUP + 3 + DUP I +
            BEGIN DUP SIZE <
            WHILE 0 OVER FLAGS + C! OVER + REPEAT
            DROP DROP 1+
         THEN
       LOOP
       .  ."PRIMES" ;
```

Listing 1.

Developing benchmark figures was interesting. I started with the classical sieve, which indicated BASCOM and TURBO 30 times better than COBOL. That's fine for integer arithmetic and heavy loop control. But I wanted to do screen formatting. I wrote a program that moved a lot of character strings, did some easy IF tests, included some editing and de-editing, and did a little "high order" arithmetic. The goal was a typical applications mix. The results changed dramatically. BASCOM slows down on strings. COBOL breaks for decimal arithmetic, TURBO chokes on editing. On balance, the finish was close: they are all 20-30 times slower than a small mainframe.

Does this mean a 60-sec response time? No. The mainframe can execute your code faster than a PC but it has more overhead in its operating system, communications controllers, access methods, and virtual memory. Moreover, benchmarks are run at night. From 8 a.m. to 5 p.m. you're lucky to get 10% of the big CPU. Ignoring file access time, response of mbp COBOL (running on an IBM PC) is usually better than—and sometimes about equal to—the same program on a host. Perhaps more important than the absolute response time is a low deviation. Studies have shown that users are upset more by erratic times than by long times.

I bought mbp because it's the best on the market today. It's a full-featured, level 2 ANS 74 COBOL with decent I/O and screen handling. I found it follows the "letter of the law" and generally the spirit, except it doesn't support COMP-3 or fractional exponents (I ** .5). There are a few minor irritations at the source level, but the biggest irritation is its size and compile speed.

Mbp COBOL takes 5-10 min to compile an average-size program and 1.5 meg on hard disk. You could compile with the floppies, but you wouldn't want to. As a heavy developer, it slows me down too much. Why is it so slow? Partly because of a slow loader (this is a multiple-pass compiler) and partly because, although it outputs machine language, the compiler itself is in nasty old p-code. Object code is big, sometimes twice as big as on the mainframe. It's written in CDL2 (an early C) in the Federal Republic of Germany.

Teaching someone the rules of a language and expecting that person to write a good program is like teaching someone to type and expecting great literature. It applies to compiler writers as well as applications programmers. Pascal compilers are written in Pascal, C compilers are written in C, but COBOL compilers are never written in COBOL. As a result, COBOL compilers are technically correct but lack a "feel" for the language.

So what's the answer? Rewrite everything in Modula 2 or True BASIC? Live with mbp? Wait for a better COBOL? If you think COBOL is over the hill then rewrite. If you like COBOL, buy mbp and wait for something better. Either way, recognize that the choice is a function of your psyche, not the quality of the language.

There is no theoretical or practical reason why COBOL should be slower or bigger than any other language. If anything, it should be faster. For example, in Listing 1 see how I initialized the array of FLAGS with a single move. That's faster than stepping through with a subscript. Some implementations (e.g., Burroughs' medium systems and CII/Bull Level 64) have produced very efficient code. A really good compiler would include one-pass compile and the X3.23 "80" standard. It should sell for under $100. I am working on such a compiler.

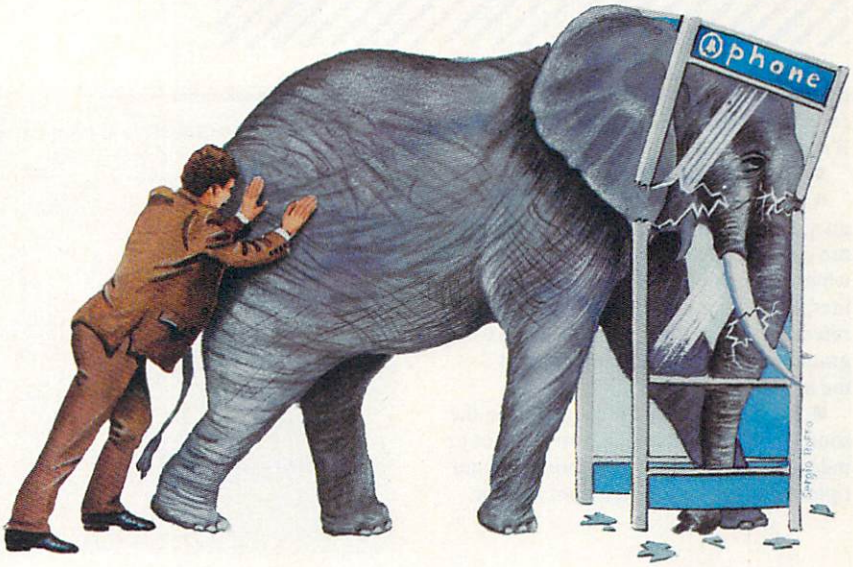
How to write a bad program

Programming is an unnatural act. As in golf, power and accuracy must come from style and not from brute force. Good style must be learned and constantly practiced. Bad programmers, like all amateurs, don't want to make the intellectual commitment. They think they can do it by intuition. As a result, they all make the same mistakes and have a common style that is easy to spot.

Here are the giveaway traits of a hacker:

- Bottom-up, rather than top-down,

Squeezing A Large Program Into A Small Memory Space?



It's time you got Plink86,™ the overlay linkage editor that's bringing modular programming to Intel 8086-based micros.

With Plink86,* you can write a program as large and complex as you want and not worry about whether it will fit within available memory constraints. You can divide your program into any number of tree-structured overlay areas. 4095 by 32 deep. Work on modules individually. Then link them into executable files. All without making changes to your source program modules.

Use the same module in different programs. Experiment with changes to the overlay structure of an existing program. Use one overlay to access code and data in other overlays.

Plink86 is a two-pass linkage editor that links modules created by the Microsoft assembler or any of Microsoft's 8086 compilers. Plink86 also works with other popular languages, like Lattice C, C86, or mbp/COBOL. And supports symbolic debugging with Phoenix' Pfix86 Plus.™

Plink86 includes its own object module library manager – Plib86™ – that lets you build libraries from scratch. Add or delete modules from existing libraries... Merge libraries... Or produce cross-reference listings.

Why squeeze any more than you have to? Plink86 by Phoenix. $395. Call (1) 800-344-7200. Or, write.

Phoenix

Phoenix Software Associates Ltd.
1420 Providence Highway Suite 260
Norwood, MA 02062
In Massachusetts (617) 769-7020

*Plink86 will run under PC DOS, MS-DOS™ or CP/M™-86.

Plink86, Pfix86 Plus and Plib86 are trademarks of Phoenix Software Associates Ltd.
MS-DOS is a trademark of Microsoft Corporation. CP/M is a trademark of Digital Research, Inc.

CIRCLE 40 ON READER SERVICE CARD

design. In a bad program, reading a record is at the top. In a good program, it is at the bottom

- Initialization is done last, not first
- Poor command of the English language. Errors in spelling and grammar are common. A hacker's description of what a program does tends to be too literal, too long, and often has too many references to hardware. A good programmer describes the function, not the mechanics
- Lack of concern for cosmetics in the source code. Failure to line up pictures or indent IF statements is common with multiple statements often on a line. Author,

TABLE 1: Comparative timings and costs of COBOL and other compilers.

Language	Cost	Relative Speed
Mainframe (.5 mips)	$4000	1
BASCOM	300	20
TURBO Pascal	50	30
mbp	850	30
BASIC interp	free	160
CIS/MicroFocus	1800	160
Ryan McFarland	600	180
IBM/Microsoft	600	200

Your PC can now "speak" C when using...

BASTOC™

A BASIC-To-C Translator

JMI's BASTOC is a versatile software tool which converts BASIC source programs to C source programs. BASTOC also translates multiple dialects of BASIC.

BASTOC FEATURES
- Supports Microsoft Disk BASIC.
- Full run-time support library is provided.
- Provides conventional BASIC compiler when used in conjunction with a C compiler.
- Produces formatted and structured C code, easily maintained and modified.
- The output of BASTOC may be used directly as input to a standard C compiler.
- Several dialects of BASIC may be translated to C on the same system.
- BASTOC will soon translate CBASIC and Applesoft BASIC.

AVAILABLE NOW

JMI's BASTOC is available now for the IBM PC and PC-compatible computers using PC-DOS or MS-DOS, and the Radio Shack TRS-80, Model 16, running XENIX. BASTOC will be available for other systems soon!

ORDER NOW

The BASTOC single unit binary price is $350, and includes documentation, media, and shipping. To order, or to get more information, write or call JMI Software Consultants, Inc., 215-657-5660. Check, Money Orders, VISA and MC are acceptable.

Registered Trademarks: IBM—International Business Machines Corp., Microsoft, MS, XENIX—Microsoft Corp.; Applesoft—Apple Computer, Inc.; TRS-80—Tandy Radio Shack Corp.; CBASIC—Digital Research, Inc.

Distributors: **Australia**, Fawnray Pty. Ltd., P.O.B. 224, Hurtsville, NSW 2220 (612) 570-6100; **Japan**, Advanced Data Controls Corp., Chiyoda-Ku, Tokyo (03) 263-0383; **United Kingdom**, Real Time Systems, Newcastle upon Tyne, 0632-733131

jmi

JMI SOFTWARE CONSULTANTS, INC.
1422 EASTON ROAD • ROSLYN, PA 19001 • 215-657-5660

CIRCLE 26 ON READER SERVICE CARD

installation information, and date are usually missing

- Fear of the machine. Hackers usually avoid language features that seem "too complicated" for the compiler—i.e, never use indexing, qualification or nested IF statements. They seldom use complex conditionals, and they handle complicated problems awkwardly
- FD's and data entries include all optional phrases and are in the same sequence given in the book
- Poor use of level numbers (e.g., 77's). A bad programmer hesitates to put a picture on an 01, and tends to use level numbers 02, 03, etc., rather than 05,10. He or she often uses *REDEFINES* frequently, for example:

```
05 FIELD1A PIC X(3).
05 FIELD1 REDEFINES
   FIELD1A PIC 9(3).
```

rather than:

```
05 FIELD1A.
   10 FIELD1 PIC 9(3).
```

- Redundant data entries instead of *OCCURS*
- *GOTO* (the mark of the beast)
- Abbreviations. It's even worse when hackers use abbreviations inconsistently. Department, for example, could be DEPT, DPT, DP or DEP, all in the same program. In a good program almost all names are written out
- Inability to distinguish between algebraic variables and character strings. To the literal mind they are all just bytes of memory. Strings are wrongly described as numeric (e.g., zip code). Numeric literals are moved to alphanumeric fields. Numeric fields are unsigned. Literals are right-filled with spaces
- Lack of concern for output cosmetics. Bad programmers do not center their headlines, and they write their titles for other programmers, not for users. They never put a program identification in the report heading, and usually do a sloppy job of date, time, and page. Hackers do printing reports with *DISPLAYS* and give inappropriate, unedited displays to the operator
- Paragraph order is temporal. *OPEN FILES* is first; *CLOSE FILES* is last. *STOP RUN* is the last program line
- Unnecessary and/or meaningless "tags" rather than paragraph names; paragraphs that are fallen into. It's worse if they are both fallen into and performed
- Flags (switches), especially a first-time flag

Some programs run like a well-oiled machine; others run like an empty footlocker falling down a flight of stairs. Listing 2 (available on the COMPUTER LANGUAGE bulletin board service: (415) 957-9370) is a bad COBOL program. Listing 2 (also available on the bulletin board) is the same program written correctly. Note how the logic telescopes and how each program has a beginning, middle, and end.

It's sad to say, but most real world programs are like Listing 3. Perhaps you got turned off to COBOL by reading or writing such a program. If Modula 2 had been abused for two decades, it would have done just as poorly.

It is tempting to fault COBOL, economics, education, employee selection or corporate dynamics, but the one really to blame is the person who wrote the bad code.

Robert Wagner is a programming manager at Furr's Inc., a regional supermarket chain headquartered in Lubbock, Texas. For the past 22 years he has written about 4 million lines of application code, mostly in structured COBOL. He is author of DYL240, *an all-time bestseller.*

Use ALL the Power of Your MS-DOS, IBM PC-DOS, or CP/M-80 System with UNIX-Style Carousel Tools

CAROUSEL TOOL KIT
CAROUSEL TOOL KIT
CAROUSEL TOOL BOOK

```
ch "CP/M" "MS-DOS" <doc >newdoc
diff newdoc doc | more
ed newdoc
kwic newdoc | sortmrg | uniq | unrot >index
make -f makdoc ndx
```

Carousel Tools and Carousel ToolKits are trademarks of Carousel MicroTools, Inc. CP/M is a trademark of Digital Research; IBM is a trademark of International Business Machines; MS is a trademark of MicroSoft; UNIX is a trademark of Bell Laboratories.

CAROUSEL TOOLS are a proven set of over 50 programs designed to be used with pipes, redirected I/O and scripts. In the style of UNIX each Tool does one thing well, and the Tools can be used together to do more complex tasks.

YOU ACCOMPLISH MORE using Carousel Tools: better programming and documentation support, simpler data and file housekeeping, more general file handling.

TOOLS FOR PC/MS-DOS 2.x AND CP/M-80 are available now. The DOS ToolKit is $149. The CP/M ToolKit is $249 and includes a *shell* to provide pipes, redirected I/O, and scripts. Source code is available for $100 more.

ORDER YOUR TOOLKIT TODAY. VISA

CALL OR WRITE:

CAROUSEL MICROTOOLS, INC.
609 Kearney Street, El Cerrito, CA 94530 (415) 528-1300

CIRCLE 7 ON READER SERVICE CARD

C

Getting Started

Examples & Functions

New version for MS-DOS includes library with IBM-PC and Z100 graphics for CI-C86 v2 compiler. Structured, indexed collection of over 70 examples and functions for learning the C programming language. Data types, declarations, preprocessor, expressions, looping, relational tests, arrays, pointers, strings, functions, macro definitions, structures, bitwise, I/O, and customization to features of specific computer systems. Library of essential source codes is applicable to most standard C compilers with little or no modifications. Manual includes hints and comments for conversion between popular compilers such as C/80, BDS-C, Aztec, DRC and CI-C86. Available in two versions:

for CP/M (Z90, Z100 - SSDD 5" Disc) $39.95
for MS-DOS (IBM-PC, Z100 - DSDD 5" Disc) . . . $39.95

Send check or money order. 6.5% tax for Calif. residents.

EIGENWARE TECHNOLOGIES
13090 LA VISTA DRIVE
SARATOGA, CA 95070
(408) 867-1184

C and Pascal Compilers
Programming Editors

Write or call for low prices.

CIRCLE 19 ON READER SERVICE CARD

THE MOST EXTENSIVE C LIBRARY ANYWHERE

FOR THE IBM AND PC XT

THE GREENLEAF FUNCTIONS
Library for C Programmers

Total Access to IBM PC and XT
Compatible with DOS 2.0, 1.1, CI C86, Lattice, and Microsoft C – Versions 1 and 2

PARTIAL CONTENTS
• DOS 2.0 - over 25 functions • Complete Video Access for Text and Graphics •Over 60 String Functions • Rainbow Series Color Text • Time and Date • Over 40 Printer Functions • Function and Special Keys • RS232 Async • All BIOS Functions • Software Diagnostics • Disk functions • Utility functions • and more . . .

THE GREENLEAF FUNCTIONS . . .
Nearly 200 functions, 220 page manual, 3 Libraries, Extensive Examples of each function, Full Source Code

$175.00
Add $7.00 for shipping.
Specify Compiler
MC/VISA Accepted
Prices subject to change without notice.
Dealer Inquiries Welcome.

(214) 446-8641

GREENLEAF SOFTWARE©

GREENLEAF SOFTWARE, INC. ◆ 2101 HICKORY DRIVE ◆ CARROLLTON, TEXAS 75006

CIRCLE 23 ON READER SERVICE CARD

Discover Forth

Join the **FORTH Interest Group**

The FORTH Interest Group (FIG) is a non-profit member-supported organization, devoted to the Forth computer language. Join our 4700+ members and discover Forth. We provide our members with the information and services they need, including:

Over fifty local FIG chapters (general and special interest) meet throughout the world on a regular basis.

Forth Dimensions magazine is published six times a year and addresses the latest Forth news. A one year subscription to FD is free with FIG membership.

The FIG-Tree is the FIG-sponsored, on-line computer data base that offers members a wealth of Forth information. Dial (415) 538-3580 using a modem and type two carriage returns.

Forth publications: a wide variety of high quality and respected Forth-related publications (listings, conference proceedings, tutorials, etc.) are available.

The FIG HOT LINE (415) 962-8653, is fully staffed to help you.

The Job Registry helps match Forth programmers with potential employers.

All this and more for only $15.00/yr. ($27.00 foreign) Just call the FIG HOT LINE or write and become a FIG member. (VISA or MC accepted.)

Don't miss our upcoming 6th Annual Forth Convention November 16-17, 1984 at the Hyatt Palo Alto in Palo Alto, CA. Call or write for details.

FORTH Interest Group

(415) 962-8653
PO Box 1105
San Carlos
CA 94070

CIRCLE 21 ON READER SERVICE CARD

COMPUTER RESOURCES *of WAIMEA*

* * * EASY TO USE * * *
Macro Programs for

Spellbinder™

We have been using and working with Spellbinder since late 1981. We use computers extensively in the day-to-day operation of our business and have developed a number of programs which we find useful. We recently formed a software development and marketing company - Computer Resources of Waimea, to promote and market these programs, most being enhancements and macro programs running under Spell binder. Spellbinder's macro programming language M-Speak is extremely versatile and in our opinion is one of the best kept "secrets" in the world of micro computers. We have a number of macro programs for the end user, a number of utilities for the programmer, and for those who want a more or less organized instruction set for M-Speak, our head programmer has compiled his personal notes into a booklet which the M-Speak user should find very useful. It can be purchased for $10.00. Send for our complete listing.

P.O. Box 1206 Kamuela, Hawaii 96743
(808) 885-7905

CIRCLE 12 ON READER SERVICE CARD

Exploring Ada and Modula-2

By Namir Clement Shammas

Since its birth in the early 1970s, Pascal has inspired many new insights and approaches toward structured program design. But as programming methods and requirements evolved, Pascal's limitations began to appear.

The birth of Ada and Modula-2 represents the evolution of Pascal into a language for real-world applications. These two languages have a good number of new features and programming concepts in common. For example, they both stress software modularity—an aspect vital to big software projects. Both languages may interact directly with a system's hardware as well as offer multitasking capabilities.

However, some interesting differences affect how one must approach a program's design in Ada vs. Modula-2. In this article, we will explore both languages from five different angles: data types, identifiers, program flow control, functions and procedures, and exceptions. I am assuming that the reader is familiar with structured programming in general and with Pascal in particular.

Data types

Modula-2 has introduced two new basic data types and a third imported from a standard module: CARDINAL, BITSET, and WORD. The CARDINAL type represents zero and all positive integers within the hardware's limit. In an 8088-based system, for example, the upper limit is 65,535. This allows twice the upper value limits of the INTEGER type because INTEGER includes negative values. BITSET is a set of integers between zero and a computer-defined upper limit. Unlike Ada, Modula-2 does not have the string type as part of its basic types; strings are regarded as arrays of characters. The WORD type represents the internal word storage of the system. It also allows type conversions.

Ada, on the other hand, has introduced the "new type," reflecting a disciplined data type checking function. The new types have the same basic data representation as the parent types but cannot be engaged with them (or any other separately declared new types) in an expression. The purpose is to forbid identifiers of a similar type from becoming intermixed when such an action would not reflect any meaningful relationship. In this situation logical expression errors are detected. Consider the example in Listing 1. If the identifier My_SSN were of type INTEGER, this expression would be compiled with no errors even though it is meaningless.

Ada allows the declaration of unconstrained array types. This means that we can define general purpose matrices. However, no identifier can be assigned to them—instead, the array bounds must be declared. Consider the unconstrained matrix type, MATRIX, and the declared identifier, MAT5, in Listing 2. Ada allows the use of INTEGER ranges for possible negative indices.

The designers of Ada have tackled the problem of numerical precision with floating point. Rather than being at the mercy of the hardware or a certain implementation, Ada offers the ability to control the precision desired. This is good news for accounting package programmers, engineers, and scientists.

Identifiers

Both languages offer array attributes that reveal information about those certain arrays needed in writing general purpose library modules. Not surprisingly, Ada offers far more attributes, allowing modules to be less error prone. These predefined attributes are clearly spelled out in Appendix 1 of *Programming in Ada*.[1]

Unlike Modula-2, Ada also allows identifiers to be initialized as they are declared. An example would be:

```
My_Phone Number : STRING :=
    ''(804) 282 - 2294'';
My_Address : STRING :=
    ''1533F Honey Grove'';
```

This feature is handy when setting default values to identifiers, and it can extend to all data types.

Ada allows some array operations, which has the net effect of shortening code length. Consider the identifier My_Address, declared previously. Suppose that I moved into a different apartment on the same street with the new number "1521A Honey Grove". Only the third, fourth, and fifth characters have changed, and Ada will allow me to carry out the one change in the following way:

```
My_Address(3..5) := ''21A'';
      -- overwrite 3rd to 5th char.
Put(My_Address);
      -- Will display
      --''121A Honey Grove''
```

Doing the above with Modula-2 is not as smooth if no string manipulation module is used.

Both Modula-2 and Ada handle temporary identifiers differently. Modula-2 has followed the Pascal concept of creating temporarily nameless identifiers that are accessed by pointers whose creation and removal is not tied to any code structure. The programmer may remove these identifiers anywhere in the program he or she sees fit, or not remove them at all.

Ada takes an entirely different approach. It allows the creation of named identifiers and ties their existence to the so-called block structure. The latter has the general structure as presented in Listing 3.

Consequently, the following rules and features are observed:

- The temporary identifiers are declared in the usual manner and with optional initial values
- The temporary identifiers will be automatically removed at the end of the block in which they were declared
- Duplicate names are allowed. However, in such cases the most recently declared identifier is the one that is "visible." It temporarily takes precedence over other identifiers with the same name. Once it is removed at the end of its block, the next most recent identifier becomes visible again, and so on (Listing 4).

Program flow control

Modula-2 has retained Pascal's *FOR-DO*, *WHILE-DO*, and *REPEAT-UNTIL* loops. Ada, on the other hand, has no counterpart for the *REPEAT-UNTIL* structure. Both languages have introduced the open loop using an *EXIT* command to avoid being trapped in a loop. Ada regards the *FOR* and *WHILE* loops as specialized loops. It allows all loops to have labels that can be used by the *EXIT* statements. This becomes very useful in the case of nested loops when determining which loops to exit. This feature allows smoother exits and easier code than Pascal or Modula-2.

Modula-2 does not implement any form of *GOTO*s (are you happy C.A.R. Hoare?). Ada, in fact, does allow the ever abusable *GOTO* with a label to indicate the program flow resumption. This seems

Introducing

WALTZ LISP T.M.

The one and only adult Lisp system for CP/M users.

Waltz Lisp is a very powerful and complete implementation of the Lisp programming language. It includes features previously available only in large Lisp systems. In fact, Waltz is substantially compatible with Franz (the Lisp running under Unix), and is similar to MacLisp. Do not be deceived by the low introductory price.

Waltz Lisp is a perfect language for Artificial Intelligence programming. It is also suitable for general applications. In fact, due to the ease of handling of textual data and random file access functions, it is often easier to write a utility program in Waltz Lisp than in any other programming language. Several general purpose utilities (including **grep** and **diff**) written entirely in Waltz Lisp are included with the interpreter.

Much faster than other microcomputer Lisps. • Long integers (up to 611 digits). Selectable radix. • True dynamic character strings. Full string operations including fast matching/extraction. • Random file access. • Binary files. • Standard CP/M devices. • Access to disk directories. • Functions of type lambda (expr), nlambda (fexpr), lexpr, macro. • Splicing and non-splicing character macros. • User control over all aspects of the interpreter. • Built-in prettyprinting and formatting facilities. • Complete set of error handling and debugging functions including user programmable processing of undefined function references. • Optional automatic loading of initialization file. • Powerful CP/M command line parsing. • Fast sorting/merging using user defined comparison predicates. • Full suite of mapping functions, iterators, etc. • Over 250 functions in total. • Extensive manual with hundreds of illustrative examples.

Waltz Lisp requires C/PM 2.0, Z80 and 48K RAM (more recommended). SS/SD 8" and most common 5" disk formats.

Introductory Price....$94.50

Manual only $20.00
(refundable with order)

additional charges
$10.00 conversion fee for 5" Diskettes
$3.00 C.O.D. charge

PRO CODE T.M.
INTERNATIONAL
P. O. Box 7301
Charlottesville, VA 22906

Call toll free 1-800-LIP-4000 Ask for Dept. #5
In Oregon and outside U.S.A. call 1-503-684-3000

Unix® Bell Laboratories. CP/M® Digital Research Corp.

CIRCLE 43 ON READER SERVICE CARD

TOTAL CONTROL:

FORTH: FOR Z-80®, 8086, 68000, and IBM® PC

Complies with the New 83-Standard

GRAPHICS • GAMES • COMMUNICATIONS • ROBOTICS DATA ACQUISITION • PROCESS CONTROL

- ***FORTH*** programs are instantly portable across the four most popular microprocessors.
- ***FORTH*** is interactive and conversational, but 20 times faster than BASIC.
- ***FORTH*** programs are highly structured, modular, easy to maintain.
- ***FORTH*** affords direct control over all interrupts, memory locations, and i/o ports.
- ***FORTH*** allows full access to DOS files and functions.
- ***FORTH*** application programs can be compiled into turnkey COM files and distributed with no license fee.
- ***FORTH*** Cross Compilers are available for ROM'ed or disk based applications on most microprocessors.

Trademarks: IBM, International Business Machines Corp.; CP/M, Digital Research Inc.; PC/Forth+ and PC/GEN, Laboratory Microsystems, Inc.

FORTH Application Development Systems include interpreter/compiler with virtual memory management and multi-tasking, assembler, full screen editor, decompiler, utilities and 200 page manual. Standard random access files used for screen storage, extensions provided for access to all operating system functions.

Z-80 FORTH for CP/M® 2.2 or MP/M II, $100.00; **8080 FORTH** for CP/M 2.2 or MP/M II, $100.00; **8086 FORTH** for CP/M-86 or MS-DOS, $100.00; **PC/FORTH** for PC-DOS, CP/M-86, or CCPM, $100.00; **68000 FORTH** for CP/M-68K, $250.00.

FORTH + Systems are 32 bit implementations that allow creation of programs as large as 1 megabyte. The entire memory address space of the 68000 or 8086/88 is supported directly.

PC FORTH +	$250.00
8086 FORTH + for CP/M-86 or MS-DOS	$250.00
68000 FORTH + for CP/M-68K	$400.00

Extension Packages available include: software floating point, cross compilers, INTEL 8087 support, AMD 9511 support, advanced color graphics, custom character sets, symbolic debugger, telecommunications, cross reference utility, B-tree file manager. *Write for brochure.*

VISA
LMI
Laboratory Microsystems Incorporated
Post Office Box 10430, Marina del Rey, CA 90295
*Phone credit card orders to **(213) 306-7412***
MasterCard

CIRCLE 27 ON READER SERVICE CARD

Six Times Faster!

Super Fast Z80 Assembly Language Development Package

Z80ASM

- Complete Zilog Mnemonic set
- Full Macro facility
- Plain English error messages
- One or two pass operation
- Over 6000 lines/minute
- Supports nested INCLUDE files
- Allows external bytes, words, and expressions (EXT1 * EXT2)
- Labels significant to 16 characters even on externals (SLR Format Only)
- Integral cross-reference
- Upper/lower case optionally significant
- Conditional assembly
- Assemble code for execution at another address (PHASE & DEPHASE)
- Generates COM, HEX, or REL files
- COM files may start at other than 100H
- REL files may be in Microsoft format or SLR format
- Separate PROG, DATA & COMMON address spaces
- Accepts symbol definitions from the console
- Flexible listing facility includes TIME and DATE in listing (CP/M Plus Only)

SLRNK

- Links any combination of SLR format and Microsoft format REL files
- One or two pass operation allows output files up to 64K
- Generates HEX or COM files
- User may specify PROG, DATA, and COMMON loading addresses
- COM may start at other than 100H
- HEX files do not fill empty address space.
- Generate inter-module cross-reference and load map
- Save symbol table to disk in REL format for use in overlay generation
- Declare entry points from console
- The FASTEST Microsoft Compatible Linker available

SPEED! SPEED! SPEED!

- Complete Package Includes: Z80ASM, SLRNK, SLRIB - Librarian and Manual for just $199.99. Manual only, $30.
- Most formats available for Z80 CP/M, CDOS, & TURBODOS
- Terms: add $3 shipping US, others $7. PA add 6% sales tax

For more information or to order, call:

1-800-833-3061

In PA, (412) 282-0864

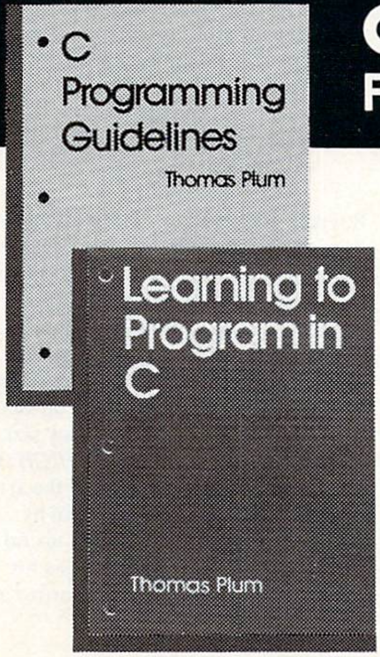
Or write: SLR SYSTEMS
1622 North Main Street, Butler, Pennsylvania 16001

SLR Systems

CIRCLE 49 ON READER SERVICE CARD

C LANGUAGE PROGRAMMING

From Plum Hall...the experts in C training



Learning to Program in C 372 pp., 7½" x 10", Price $25.00
A practical, step-by-step guide for everyone acquainted with computers who wants to master this powerful "implementer's language". Inside, you will learn how to write portable programs for the full spectrum of processors, micro, mini and mainframe

C Programming Guidelines 140 pp., 7½" x 10", Price $25.00
A compilation of standards for consistant style and usage of C language. Arranged in manual page format for easy reference, it presents time-tested rules for program readability and portability.

FREE
C LANGUAGE POCKET GUIDE!

A handy C language programming pocket guide is yours *free* when you order either (or both) of the manuals above. A full 14 pages of valuable C language information!

PLUM HALL
1 Spruce Av, Cardiff NJ 08232
The experts in C and UNIX™ training.
Phone orders: 609-927-3770

Please send me: ________ information on C and UNIX Training Seminars
________ copies of Learning to Program in C @ $25.00/copy
________ copies of C Programming Guidelines @ $25.00/copy
NJ residents add 6% sales tax.

NAME ____________________
COMPANY ____________________
ADDRESS ____________________
CITY/STATE/ZIP ____________________
☐ Check ☐ American Express ☐ Master Card ☐ Visa
CARD # ____________ EXP. DATE ____ / ____ Signature ____________
UNIX is a trademark of AT&T Bell Laboratories

CIRCLE 41 ON READER SERVICE CARD

DeSmet C

The fastest 8088 C Compiler available

FULL DEVELOPMENT PACKAGE
- C Compiler
- Assembler
- Linker and Librarian
- Full-Screen Editor
- Newsletter for bugs/updates

SYMBOLIC DEBUGGER
- Monitor and change variables by name using C expressions
- Multi-Screen support for debugging PC graphics and interactive systems
- Optionally display C source during execution
- Breakpoint by Function and Line #

COMPLETE IMPLEMENTATION
- **Both** 1.0 **and** 2.0 DOS support
- Everything in K&R (incl. STDIO)
- Intel assembler mnemonics
- **Both** 8087 **and** Software Floating Point

OUTSTANDING PERFORMANCE

Sieve Benchmark

COMPILE	4 Sec. RAM —
	22 Sec FDISK
LINK	6 Sec. RAM —
	34 Sec. FDISK
RUN	12 Sec.
SIZE	8192 bytes

DeSmet C Development Package $159

To Order Specify:

Machine ____________

OS ☐ MS-DOS ☐ CP/M-86

Disk ☐ 8" ☐ 5¼ SS ☐ 5¼ DS

C WARE CORPORATION
P.O. BOX 710097
San Jose, CA 95171-0097
(408) 736-6905

California residents add sales tax. Shipping: U.S. no charge, Canada add $5, elsewhere add $15. Checks must be on a US Bank and in US Dollars.

NGS FORTH

A FAST FORTH OPTIMIZED FOR THE IBM PERSONAL COMPUTER AND MSDOS COMPATIBLES.

*79 STANDARD

*FIG LOOKALIKE MODE

*PC-DOS COMPATIBLE

*ON-LINE CONFIGURABLE

*ENVIRONMENT SAVE & LOAD

*MULTI-SEGMENTED

*EXTENDED ADDRESSING

*AUTO LOAD SCREEN BOOT

*LINE AND SCREEN EDITORS

*DECOMPILER & DEBUGGING AIDS

*8088 ASSEMBLER

*BASIC GRAPHICS & SOUND

*NGS ENHANCEMENTS

*DETAILED MANUAL

*INEXPENSIVE UPGRADES

*NGS USER NEWSLETTER

A COMPLETE FORTH DEVELOPMENT SYSTEM.

PRICE: $70

PLEASE INCLUDE $2 POSTAGE & HANDLING WITH EACH ORDER.
CALIFORNIA RESIDENTS : INCLUDE 6.5% SALES TAX.

NEXT GENERATION SYSTEMS
P.O.BOX 2987
SANTA CLARA, CA. 95055
(408) 241-5909

to be useful in handling errors that occur inside well-nested loops, allowing easier code to be written.

Functions and procedures

Modula-2 sees functions merely as procedures that return a value. Thus the keyword FUNCTION has been dropped. There are some important differences in using procedures and functions that can affect code writing.

Overloading is an Ada feature. The language designers forsaw certain conflicts arising from having software teams co-develop separate modules. Among the problems was the repetitive use of the same procedure or function name by different programmers to perform different jobs. One possible solution, which Modula-2 also offers, is to use the imported procedure name preceeded by the module name:

```
Floatlo.Put(X); -- Procedure Put from
      -- module Floatlo
Textlo.Put("HELLO");
      -- Procedure Put from
      -- module Textlo
```

To make things easier, Ada even allows you to drop the source module name and still create no conflict. The only condition is that the argument call lists must not be identical. Ada sees the parameter list as a complementary part (with the routine name) of the routine's identity. Thus one can use the procedure *Put* repetitively to declare procedures that display data of different types as in:

```
Procedure Put(X : Real);
    -- output floating point
Procedure Put(I : Integer);
    -- output integer
Procedure Put(S : String);
    -- output String
```

But Ada goes even further. It allows the overloading of operators. This enables a programmer to write routines to add, subtract, and multiply matrices.

Modula-2 has introduced its own limited implementation of unconstrained arrays by using the ARRAY OF keyword. The function in Listing 5 sums up an entire one dimensional array of any size. In this example we are using the *HIGH* function to detect the upper limit of the array while the lower one is set to zero by Modula-2. This feature is very useful in writing procedures and functions for strings when quoted text is regarded as an array of characters. "Modula-2: No strings attached!"[2] has a string module that uses this feature extensively. Consider the following procedural call:

```
PROCEDURE WriteString(S :
      ARRAY OF CHAR);
      (* Displays a string variable or a
      quote *)
```

which can be called to display a message,

```
WriteString('press any key');
```

The limitation of this unconstrained array feature is that it is restricted to one dimensional arrays. Thus, writing general purpose routines for matrix operation is not as straightforward. R. Weiner suggests the use of ALLOCATE and DEALLOCATE to create matrices as temporary identifiers.[3]

Another method is to "unwrap" the matrix into one long array and to use two identifiers to record the number of rows and columns. One can even include a function that will simulate two or more dimensions. Another Modula-2 feature is the ability to pass procedure names in argument calls. The latter must be declared as procedure types.

Consider the following example. Suppose I would like to write a function that performs numerical integration on users' functions. I start by declaring *UserFunc* as a procedure type for users' function,

```
TYPE UserFunc = PROCEDURE
      (REAL): REAL;
```

and then declare the variable *MyFunc*

```
MyFunc : UserFunc;
```

Somewhere in my program I will assign a function, say *sqrt*, to *MyFunc*

```
MyFunc := sqrt (* sqrt is imported
      from module Mathlib0 *)
```

and would be able to obtain the area under the curve by calling my *Area* function to integrate from zero to one at 0.1 increments:

```
Result := Area(0.0,1.0,0.1,MyFunc);
```

The definition of the *Area* function would begin with

```
PROCEDURE
            Area(Low,High,Incr : REAL;
         Myfunc : UserFunc): REAL;
```

Ada allows programmers to develop generic procedures and functions. Code templates can help the programmer focus on a general mechanism that deals with abstract data types. Operations such as stack manipulation, lists management, sorting, and searching can be performed on a variety of data types. You can sort integers, reals, and strings, but the algorithm used is essentially the same. You can manipulate a stack of numbers and a stack of names, yet the basic operations are the same (push, pop, rotate, exchange, etc.). This is where Ada's generic concept comes in handy. The generic code would contain algorithms applicable to a wide variety of data types and thus involve private data types.

To use generic functions, the newly customized function or procedure would have to specify the data type involved. Consider the following example where a generic procedure *ROTATE* will swap two data items (Listing 6). When a programmer wants to use the program in Listing 6 to swap integers using a procedure names *ROTATE_INTGRS* the following line must be present,

```
procedure ROTATE_INTGRS is new
      ROTATE(INTEGER);
```

which will create an operating procedure from the generic template dealing with integer types only.

Modula-2 can emulate generics by us-

For your IBM/PC

mbp COBOL: 4 times faster, and now with SORT & CHAIN.

$750.

mbp COBOL can be summed up in one word: fast.

Because it generates native machine language object code, the mbp COBOL Compiler executes IBM/PC* programs *at least* 4 times faster (see chart).

GIBSON MIX Benchmark Results
Calculated S-Profile
(Representative COBOL statement mix)

Execution time ratio

mbp COBOL	Level II** COBOL	R-M*** COBOL	Microsoft**** COBOL
1.00	4.08	5.98	6.18

128K system with hard disk required. *IBM/PC is an IBM TM; **Level II is a Micro Focus TM; ***A Ryan-McFarland TM; ****A Microsoft TM.

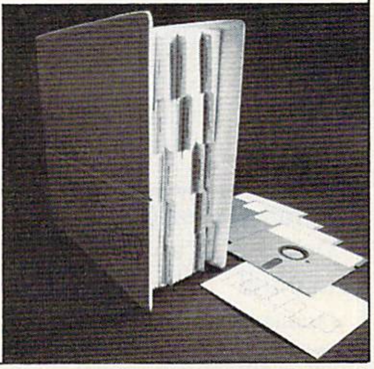
Fast also describes our **new SORT,** which can sort four-thousand 128-byte records in less than 30 seconds. A callable subroutine or stand-alone, 9 SORT control fields can be specified. And our **new CHAIN** is both fast and secure, conveniently transferring control from one program to another, passing 255 parameters. Plus, **new extensions** to ACCEPT & DISPLAY verbs give better, faster interactive programming.

The complete COBOL. An Interactive Symbolic Debug Package included standard; Multi-Keyed ISAM Structure; listing options allow source & object code, map & cross-reference checking; GSA Certification to ANSI '74 Level II; mbp has it all.

It's no surprise companies like Bechtel, Chase, Citicorp, Connecticut Mutual, and Sikorsky choose mbp COBOL; make it your choice, too. mbp is available at Vanpak Software Centers, or direct. For complete information, write **mbp Software & Systems Technology, Inc.,** 7700 Edgewater Drive, Suite 360, Oakland, CA 94621, or phone 415/632-1555 —today.

mbp®

CIRCLE 31 ON READER SERVICE CARD



```
Type Social_Security_Number is new INTEGER;
My_SSN : Social_Security_Number;
Part_Number_Inventory : INTEGER;
-- The expression below is meaningless and  WRONG!
-- no logical connection between the two identifiers
Part_Number_Inventory := Part_Number_Inventory - My_SSN;
```

Listing 1.

```
Type MATRIX is array (POSITIVE RANGE <>, POSITIVE range <>)
                                                   of INTEGER;

      MAT5 : MATRIX(1..5,1..5); -- 5 by 5 square matrix
```

Listing 2.

```
declare -- block begins here
        -- new temporary identifiers are declared here
begin
        -- code statements
end;    -- block ends here.  All identifiers declared
        -- at the beginning of the block are removed
```

Listing 3.

C Advanced Screen Management
Made Easy

WINDOWS
FOR C™

FOR THE IBM PC
+ COMPATIBLES
Lattice C, DeSmet C
C86, Microsoft C
All versions

ADVANCED FEATURES

- Complete window system
- Unlimited windows and text files
- Nest and overlap windows
- Overlay, restore, and save windows
- Horizontal and vertical scrolling
- Word wrap, auto scroll
- Print windows
- Highlighting
- Fast screen changes
- No snow, no flicker

WINDOWS++

Much more than a window display system. **Windows for C** is a video display toolkit that simplifies all screen management tasks.

SIMPLIFY • IMPROVE

- Menus
- Help files
- Data screens
- Editors

ALL DISPLAYS

C SOURCE MODULES FOR

pop-up menus, multiple window displays, label printer, cursor control, text mode bar graphs.

complete building block subroutines

DESIGNED FOR PORTABILITY

*Minimal dependence on IBM BIOS and 8086 ASM

FULL SOURCE AVAILABLE
NO ROYALTIES

Windows for C	$150
Demo disk and manual	$ 30
(applies toward purchase)	

A PROFESSIONAL SOFTWARE TOOL FROM
CREATIVE SOLUTIONS
21 Elm Ave., Box T5, Richford, VT 05476

802-848-7738
Master Card & Visa Accepted
Shipping $2.50
VT residents add 4% tax

CIRCLE 13 ON READER SERVICE CARD

THE FORTH SOURCE™

MVP-FORTH

Stable - Transportable - Public Domain - Tools

You need two primary features in a software development package . . . a stable operating system and the ability to move programs easily and quickly to a variety of computers. MVP-FORTH gives you both these features and many extras. This public domain product includes an editor, FORTH assembler, tools, utilities and the vocabulary for the best selling book "Starting FORTH". The Programmer's Kit provides a complete FORTH for a number of computers. Other MVP-FORTH products will simplify the development of your applications.

MVP Books - A Series

- ☐ **Volume 1,** *All about FORTH* by Haydon. MVP-FORTH glossary with cross references to fig-FORTH, *Starting FORTH* and FORTH-79 Standard. 2nd Ed. $25
- ☐ **Volume 2,** *MVP-FORTH Assembly Source Code.* Includes CP/M®, IBM-PC®, and APPLE® listing for kernel $20
- ☐ **Volume 3,** *Floating Point Glossary* by Springer $10
- ☐ **Volume 4,** *Expert System* with source code by Park $25
- ☐ **Volume 5,** *File Management System* with interrupt security by Moreton $25

MVP-FORTH Software - A Transportable FORTH

- NEW ☐ **MVP-FORTH Programmer's Kit** including disk, documentation, Volumes 1 & 2 of MVP-FORTH Series (*All About FORTH*, 2nd Ed. & Assembly Source Code), and *Starting FORTH*. Specify ☐ CP/M, ☐ CP/M 86, ☐ CP/M+, ☐ APPLE, ☐ IBM PC, ☐ MS-DOS, ☐ Osborne, ☐ Kaypro, ☐ H89/Z89, ☐ Z100, ☐ TI-PC, ☐ MicroDecisions, ☐ Northstar, ☐ Compupro, ☐ Cromenco, ☐ DEC Rainbow, ☐ NEC 8201, ☐ TRS-80/100 $150
- ☐ **MVP-FORTH Cross Compiler** for CP/M Programmer's Kit. Generates headerless code for ROM or target CPU $300
- ☐ **MVP-FORTH Meta Compiler** for CP/M Programmer's kit. Use for applicatons on CP/M based computer. Includes public domain source $150
- ☐ **MVP-FORTH Fast Floating Point** Includes 9511 math chip on board with disks, documentation and enhanced virtual MVP-FORTH for Apple II, II+, and IIe. $450
- ☐ **MVP-FORTH Programming Aids** for CP/M, IBM or APPLE Programmer's Kit. Extremely useful tool for decompiling, callfinding, and translating. $200
- ☐ **MVP-FORTH PADS (Professional Application Development System)** for IBM PC, XT or PCjr or Apple II, II+ or IIe. An integrated system for customizing your FORTH programs and applications. The editor includes a bi-directional string search and is a word processor specially designed for fast development. PADS has almost triple the compile speed of most FORTH's and provides fast debugging techniques. Minimum size target systems are easy with or without heads. Virtual overlays can be compiled in object code. PADS is a true professional development system. Specify Computer. $500
- NEW ☐ **MVP-FORTH Floating Point & Matrix Math** for IBM or Apple $85
- NEW ☐ **MVP-FORTH Graphics Extension** for IBM or Apple $65
- NEW ☐ **MVP-FORTH MS-DOS** file interface for IBM PC PADS $80
- NEW ☐ **MVP-FORTH Expert System** for development of knowledge-based programs for Apple, IBM, or CP/M. $100

FORTH CROSS COMPILERS Allow extending, modifying and compiling for speed and memory savings; can also produce ROMable code. Specify CP/M, 8086, 68000, IBM, Z80, or Apple II, II+ $300

FORTH COMPUTER

- ☐ **Jupiter Ace** $150

Ordering Information: Check, Money Order (payable to MOUNTAIN VIEW PRESS, INC.), VISA, MasterCard, American Express. COD's $5 extra. Minimum order $15. No billing or unpaid PO's. California residents add sales tax. Shipping costs in US included in price. Foreign orders, pay in US funds on US bank, include for handling and shipping by Air: $5 for each item under $25, $10 for each item between $25 and $99 and $20 for each item over $100. All prices and products subject to change or withdrawal without notice. Single system and/or single user license agreement required on some products.

FORTH DISKS

FORTH with editor, assembler, and manual.

- NEW ☐ **APPLE** by MM, 83 $100
- ☐ **APPLE** by Kuntze $90
- ☐ **ATARI**® valFORTH $60
- NEW ☐ **CP/M**® by MM, 83 $100
- ☐ **HP-85** by Lange $90
- ☐ **HP-75** by Cassady $150
- NEW ☐ **IBM-PC**® by LM, 83 $100
- ☐ **NOVA** by CCI 8" DS/DD $175
- ☐ **Z80** by LM, 83 NEW $100
- ☐ **8086/88** by LM, 83 NEW $100
- ☐ **68000** by LM, 83 NEW $250
- ☐ **VIC FORTH** by HES, VIC20 cartridge $50
- ☐ **C64** by HES Commodore 64 cartridge $60
- ☐ **Timex** by HW $25

Enhanced FORTH with: F-Floating Point, G-Graphics, T-Tutorial, S-Stand Alone, M-Math Chip Support, MT-Multi-Tasking, X-Other Extras, 79-FORTH-79, 83-FORTH-83.

- NEW ☐ **APPLE** by MM, F, G, & 83 $180
- ☐ **ATARI** by PNS, F,G, & X. $90
- NEW ☐ **CP/M** by MM, F & 83 $140
- ☐ **Apple, GraFORTH** by I $75
- ☐ **Multi-Tasking FORTH** by SL, CP/M, X & 79 $395
- ☐ **TRS-80/I or III** by MMS F, X, & 79 $130
- ☐ **Timex** by FD, tape G,X, & 79 $45
- ☐ **Victor 9000** by DE,G,X $150
- NEW ☐ **C64** by ParSec. MVP, F, 79, G & X $96
- ☐ **FDOS** for Atari FORTH's $40
- ☐ **Extensions** for LM Specify IBM, Z80, or 8086
  - ☐ Software Floating Point $100
  - ☐ 8087 Support (IBM-PC or 8086) $100
  - ☐ 9511 Support (Z80 or 8086) $100
  - ☐ Color Graphics (IBM-PC) $100
  - ☐ Data Base Management $200
- ☐ **fig-FORTH Programming Aids** for decompiling, callfinding, and translating. CP/M, IBM-PC, Z80, or Apple $200

FORTH MANUALS, GUIDES & DOCUMENTS

- ☐ **ALL ABOUT FORTH** by Haydon. See above. $25
- ☐ **FORTH Encyclopedia** by Derick & Baker $25
- NEW ☐ **The Complete FORTH** by Winfield $16
- ☐ **Understanding FORTH** by Reymann $3
- ☐ **FORTH Fundamentals,** Vol. I by McCabe $16
- NEW ☐ **FORTH Fundamentals,** Vol. II by McCabe $13
- NEW ☐ **FORTH Tools, Vol.1** by Anderson & Tracy $20
- ☐ **Beginning FORTH** by Chirlian $17
- ☐ **FORTH Encyclopedia Pocket Guide** $7
- ☐ **And So FORTH** by Huang. A college level text. $25
- ☐ **FORTH Programming** by Scanlon $17
- ☐ **FORTH on the ATARI** by E. Floegel $8
- ☐ **Starting FORTH** by Brodie. Best instructional manual available. (soft cover) $18
- ☐ **Starting FORTH** (hard cover) $23
- ☐ **68000** fig-Forth with assembler $20
- ☐ **Jupiter** ACE Manual by Vickers $15
- ☐ **1980 FORML Proc.** $25
- ☐ **1981 FORML Proc** 2 Vol $40
- ☐ **1982 FORML Proc.** $25
- ☐ **1981 Rochester FORTH Proc.** $25
- ☐ **1982 Rochester FORTH Proc.** $25
- ☐ **1983 Rochester FORTH Proc.** $25
- ☐ **A Bibliography of FORTH References, 1st. Ed.** $15
- **The Journal of FORTH Application & Research**
  - ☐ **Vol. 1, No. 1** $20
  - NEW ☐ **Vol. 1, No. 2** $20
- ☐ **A FORTH Primer** $25
- ☐ **Threaded Interpretive Languages** $23
- ☐ **METAFORTH** by Cassady $30
- ☐ **Systems Guide to fig-FORTH** $25
- ☐ **Invitation to FORTH** $20
- ☐ **PDP-11 User Man.** $20
- ☐ **FORTH-83 Standard** $15
- ☐ **FORTH-79 Standard** $15
- ☐ **FORTH-79 Standard Conversion** $10
- ☐ **Tiny Pascal fig-FORTH** $10
- ☐ **NOVA fig-FORTH** by CCI Source Listing $25
- ☐ **NOVA** by CCI User's Manual $25

☐ **Installation Manual for fig-FORTH,** $15

Source Listings of fig-FORTH, for specific CPU's and computers. The Installation Manual is required for implementation. Each $15

☐ 1802 ☐ 6502 ☐ 6800 ☐ AlphaMicro
☐ 8080 ☐ 8086/88 ☐ 9900 ☐ APPLE II
☐ PACE ☐ 6809 ☐ NOVA ☐ PDP-11/LSI-11
☐ 68000 ☐ Eclipse ☐ VAX ☐ Z80 ☐ IBM NEW

MOUNTAIN VIEW PRESS, INC.

PO BOX 4656
MOUNTAIN VIEW, CA 94040
(415) 961-4103

CIRCLE 35 ON READER SERVICE CARD

```
declare                       -- first block declared
                              -- new identifiers declared
     I : INTEGER := 1;
begin
     Put(I);                  -- display 1
     declare                  -- second block declared
          I : Integer := 10;  -- duplicate I is declared
                              -- it becomes visible inside
                              -- the second block
          begin
               Put(I);        -- displays 10
          end;                -- end of second block
      Put(I);                 -- displays 1 again
                              -- end of outer block
end;
```

Listing 4.

```
PROCEDURE SumVector(X : ARRAY OF REAL):  REAL;
VAR i : CARDINAL;
    Sum : REAL;
BEGIN
     Sum := 0.0;
     FOR i := 0 TO HIGH(X) DO
          Sum := Sum + X[i]
     END;
RETURN Sum;
END SumVector;
```

Listing 5.

THE PROGRAMMER'S SHOP

helps compare, evaluate, find products. Straight answers for serious programmers.

SERVICES

- Programmer's Referral List
- Compare Products
- Help find a Publisher
- Evaluation Literature free
- BULLETIN BOARD - 7 PM to 7 AM 617-461-0174
- Dealer's Inquire
- Newsletter
- Rush Order
- Over 300 products

Our Free Report: PRODUCTIVITY - MSDOS

Assume use of compiler and typical editor. What commercial or public domain products, what techniques improve productivity? "Productivity with MSDOS" is a growing document with some answers. Call to request it. **Help improve it. Earn $50** credit toward any purchase when we add any description, code, or idea received from you.

RECENT DISCOVERIES

PROFILER - Examine MSDOS program execution speeds. Determine where to improve programs in any Microsoft language, Lattice, or C86. Make histograms that show time spent in portions of your program, and doing MSDOS I/O, etc. $175.

"C" LANGUAGE		LIST PRICE	OUR PRICE
APPLE:	AZTEC C - Full, ASM	$199	call
8080:	BDS C - Fast, popular	150	125
8080:	AZTEC C - Full	199	call
Z80:	ECOSOFT - Fast, Full	250	225
8086:	C86 - optimizer, Meg	395	call
8086:	Lattice - New 1.1 & 2.0	500	call
Microsoft (Lattice)	MSDOS	500	call
Digital Research - Megabyte	8086	350	269
Desmet by CWare - Fast	8086	109	99

BASIC	ENVIRONMENT		
Active Trace - debug	8080/86	$ 80	72
MBASIC-80 - MicroSoft	8080	375	255
BASCOM-86 - MicroSoft	8085	395	279
CB-86 - DRI	CPM86	600	439
Prof. BASIC Compiler	PCDOS	345	325
BASIC Dev't System	PCDOS	79	72

FEATURES

C INTERPRETERS for MSDOS - Ask about one for beginners for $85 or full development for $500.

C HELPER includes source in C for MSDOS or CPM80 for a DIFF, GREP, Flowcharter, C Beautifier and others. Manage your source code easier. $125.

PROLOG86 Interpreter for MSDOS includes tutorials, reference and good examples. Learn in first few hours. For Prototyping, Natural Language or AI. $125.

EDITORS Programming			
C Screen with source	8080/86	NA	60
EDIX - clean	PCDOS	195	149
FINAL WORD - for manuals	8080/86	300	215
MINCE - like EMACS	CPM, PCDOS	175	149
PMATE - powerful	CPM	195	175
	8086	225	195
VEDIT - full, liked	CPM, PCDOS	150	119
	8086	200	159

FORTRAN			
MS FORTRAN-86 - Meg	MSDOS	$350	$255
SS FORTRAN - 86	CPM-86	425	345
FORTRAN-80 - 66 decent	CPM-80	500	350
INTEL FORTRAN - 86	IBM PC	NA	1400
DR FORTRAN COMING			
RM FORTRAN COMING			

LANGUAGE LIBRARIES			
C to dBASE interface	8080/85	$125	$115
C Tools 1 - String, Screen	PCDOS	NA	115
C Tools 2 - OS Interface	PCDOS	NA	92
FLOAT 87 - Lattice, PL1	PCDOS	NA	115
GRAPHICS: GSX - 80	CPM80	NA	75
HALO - fast, full	PCDOS	200	175
Greanleaf for C - full, 200+.	PCDOS	NA	165
ISAM: Access Manager - 86	8086	400	300
BTRIEVE - many languages	PCDOS	245	215
PHACT - with C	PCDOS	NA	250
FABS	CPM80	150	135
PASCAL TOOLS - Blaise	PCDOS	NA	115
SCREEN: Display Mgr. 86	8086	500	375
PANEL-86 - many languages	PCDOS	295	245
WINDOWS for C	PCDOS	NA	115
Virtual Screen - Amber	PCDOS	295	call

PASCAL	ENVIRONMENT	LIST PRICE	OUR PRICE
PASCAL MT + 86	CPM86/IBM	$400	$289
without SPP	CPM80	350	249
MS PASCAL 86	MSDOS	350	255
SBB PASCAL-great, fast	PCDOS	350	315
PASCAL 64 - nearly full	COM 64	99	89
SBB Jr - best to learn	PCDOS	NA	95

OTHER PRODUCTS			
Assembler & Tools - DRI	8086	200	159
COBOL - Level II	8086	1600	1275
CODESMITH-86 - debug	PCDOS	149	139
GC LISP	PCDOS	495	475
IQ LISP - full 1000K RAM	PCDOS	175	call
Janus ADA - solid value	PCDOS	500	449
MBP Cobol-86 - fast	8086	750	695
Microshell improve CPM	8080	150	125
Microsoft MASM-86	MSDOS	100	85
PL/1-86	8086	750	560
PLINK-86 - overlays	8086	350	315
POWER - recover files	8080/86	169	139
READ CPM86 from PCDOS	PCDOS	NA	55
READ PCDOS on an IBM PC	CPM86	NA	55
Trace 86	PCDOS	125	115

Call for a catalog, literature, and answers

800-421-8006

THE PROGRAMMER'S SHOP™

128-L Rockland Street, Hanover, MA 02339.

Visa 617-826-7531, Mass: 800-442-8070 MasterCard

Note: All prices subject to change without notice. Mention this ad. Some prices are specials.

Ask about COD and POs.
All formats available.

CIRCLE 44 ON READER SERVICE CARD

```
generic
     type OBJECT is private;
     procedure ROTATE(FIRST, SECOND: in out OBJECT);

     procedure ROTATE(FIRST, SECOND:  in out OBJECT) is
     USED : OBJECT;
     begin
          USED := FIRST
          FIRST := SECOND
          SECOND := USED;
     end;
```

Listing 6.

QUALITY SOFTWARE AT REASONABLE PRICES

CP/M Software by

Poor Person Software

Poor Person's Spooler **$49.95**
All the function of a hardware print buffer at a fraction of the cost. Keyboard control. Spools and prints simultaneously.

Poor Person's Spread Sheet **$29.95**
Flexible screen formats and BASIC-like language. Preprogrammed applications include Real Estate Evaluation.

Poor Person's Spelling Checker **$29.95**
Simple and fast! 33,000 word dictionary. Checks any CP/M text file.

aMAZEing Game **$29.95**
Arcade action for CP/M! Evade goblins and collect treasure.

Crossword Game **$39.95**
Teach spelling and build vocabulary. Fun and challenging.

Mailing Label Printer **$29.95**
Select and print labels in many formats.

Window System **$29.95**
Application control of independent virtual screens.

All products require 56k CP/M 2.2 and are available on 8" IBM and 5" Northstar formats, other 5" formats add $5 handling charge. California residents include sales tax.

Poor Person Software
3721 Starr King Circle
Palo Alto, CA 94306
tel 415-493-3735

CP/M is a registered trademark of Digital Research

CIRCLE 42 ON READER SERVICE CARD

WORKMAN & ASSOCIATES
USER-FRIENDLY SOFTWARE

WRITE

The Writer's Really Incredible Text Editor lives up to its name! It's designed for creative and report writing and carefully protects your text. Includes many features missing from WordStar, such as assorted directory listings, fast scrolling, and trial printing to the screen. All editing commands are single-letter and easily changed. Detailed manual included. WRITE is $239.00.

A PRIMER ON PASCAL
FOR CP/M SYSTEMS

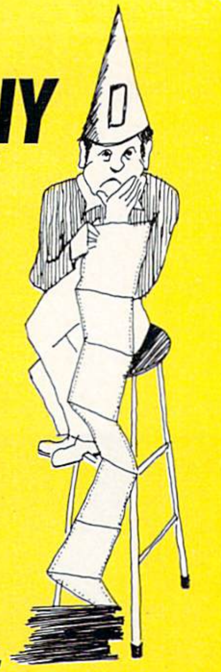
If you want to learn Pascal/M or MT+ on your own, this is the way. The Primer contains sample programs, suggestions, and notes on disk and in the manual. Now only $64.50!

WORKMAN & ASSOCIATES
112 Marion Avenue
Pasadena, CA 91106
(818) 796-4401

Disk formats include: 8", Apple, Osborne, Xerox, KayPro, Morrow, and Otrona. Please request our new catalog.

CIRCLE 56 ON READER SERVICE CARD

WHY JOHNNY CAN'T READ HIS OWN CODE



Johnny's A Good Programmer, Even Brilliant, But— Johnny works in 8080/Z80 assembly language, with a conventional assembler. That can make yesterday's brilliance today's garble, a maze of mnemonics and a jumble of meaningless labels. Johnny's program is less than self-explanatory—even for Johnny.

Johnny *could* read his own code if he used SMAL/80—the *superassembler*—and so can you. SMAL/80 boosts your program's clarity and your productivity by giving you:

■ Familiar algebraic notation in place of cryptic mnemonics—"A=A–3" for example, instead of "SUI 3" (if you know BASIC or Pascal, you *already* know SMAL/80)

■ Control structures like BEGIN...END, LOOP...REPEAT WHILE, and IF...THEN...ELSE... to replace tangled branches and arbitrary label names (eliminating up to 90% of labels with *no* overhead imposed)

■ Complete control over your processor—because SMAL/80 is a true assembler, it doesn't reduce execution speed or burden your program with its own runtime routines.

SMAL/80, the assembler that handles like a high-level language, lets you do it right the first time, and lets you read and understand your work afterward—the next day or a year later. Users say SMAL/80 has doubled and even tripled their output of quality code. But don't take our word for it—***TRY IT!***

Use SMAL/80 for 30 days. If you're not *completely* satisfied with it—for *any* reason—return the package for a full refund.

SPECIAL BONUS: Order before Sept. 30, 1984, and get *Structured Microprocessor Programming*—a $25 book ***FREE!***

SMAL/80 for CP/M-80 systems (*all* CP/M disk formats available—please specify); produces 8080/8085 and Z80 code. Now supports Microsoft .REL.**ONLY $149.95**

SMAL/80 for CP/M-80 systems, 8080/8085 output only. **SAVE $20: $129.95**

NEW! SMAL/80X65—for Apple II and IIe (requires Z80 card and CP/M); produces Z80 *and* 6502 object code. **$169.95**

Mastercard
Visa
C.O.D.'s

SMAL/80
CHROMOD ASSOCIATES
(201) 653-7615
1030 Park Ave. Hoboken, N.J. 07030

We pay shipping on prepaid orders

ing the WORD data type and specifically the ARRAY OF WORD in generic function calls. This array can accept any identifier. R. Wiener ^{4} gives a good example of this technique by presenting a generic heap and exchange sort program in Modula-2.

Exceptions

Ada was designed, following the U.S. Department of Defense's requirements, to be implemented in real-world systems. Thus software developers must be ready to handle errors that can cause the system to crash.

Consider the case where an Ada software system is implemented in a military airplane. Imagine that an electronic thermometer starts sending erroneous negative readings (in absolute degrees Rankine!). This may cause the system to crash and prevent the pilot from being able to eject.

The idea behind exceptions is to handle error events. Ada has four predefined exceptions:

■ CONSTRAINED_ERROR—Handles out of range error

■ NUMERIC_ERROR—Deals with mathematical operation errors

■ PROGRAM_ERROR—For run-time errors

■ STORAGE_ERROR—Deals with out-of-memory error.

Ada also allows the programmer to declare other exceptions. To activate an exception one uses the RAISE keyword followed by the exception name. This would cause the program flow to resume at the exception handling code portion. This portion has the list of error types and the action taken for each. In Modula-2, on the other hand, error handling is done without special jumps. This can be frustrating especially when an error is inside a deeply nested loop.

The context in which Modula-2 and Ada are viewed determines their relative strengths and weaknesses. But the programming techniques of these two highly structured languages offer tools to programmers in any context.

References

1. Barnes, J. *Programming in Ada*, Sec. Ed., Addison-Wesley.
2. Shammas, N. 1984. "Modula-2: No Strings Attached!" *Journal of Pascal, Ada, and Modula-2.* 3:2.
3. Wiener, R. 1983. "Dynamic Multidimensional Arrays in Modula-2." *Journal of Pascal, Ada, and Modula-2.* 2:6.
4. Wiener, R. 1984. "Generic Sorting in Modula-2." *Journal of Pascal, Ada, and Modula-2.* 3:1.

Namir Shammas is president of his own company, Pyramid Software. He is a member of the AdaTec chapter of the Association of Computing Machinery. He is also involved in the IEEE Computer Society and the American Statistician Society. He holds a BSC and MSC in Chemical Engineering and enjoys technical writing and programming.

ADVERTISE in the October issue of COMPUTER LANGUAGE

Reservation Deadline:
August 6th

Contact:
Carl Landau
Computer Language
131 Townsend Street
San Francisco, CA 94107
(415) 957-9353

CIRCLE 8 ON READER SERVICE CARD

PUBLIC DOMAIN SOFTWARE REVIEW

By Tim Parker

To many readers of *COMPUTER LANGUAGE* it's no surprise that public domain software continues to be a valuable source for all kinds of interesting and virtually free software programs.

For every commercial software package—be it a word processor, language assembler, or utility—there is a similar product sitting unused on disk somewhere practically free for the asking. In many cases, the public domain material is as good as or better than the commercial product.

But, naturally, this is not always true. Powerful languages, data bases, and word processors are usually commercial because of their money-making potential. To find an equivalent in the no-cost world of public domain material may be asking too much. But as far as utilities are concerned, public domain products usually set the standard.

What exactly is public domain software?

Usually an author who decides not to commercially market a program feels—for one or many reasons—that the program should be distributed to any interested parties at a minimal cost. The motivation is usually a desire to make a useful product readily accessible to other programmers who have similar interests. But sometimes it is a lack of motivation to actually market the material, perhaps due to other interests or a lack of what many call "business drive".

Whatever the reason, the usual condition attached to the released software is that it cannot be sold for profit by anyone else. (Often routines can be incorporated into commercial programs, but this is at the author's discretion.)

Public domain material does have its problems. In many cases, the programs are uninspiring, repetitious, or unworkable. But even so, many programs deserve a place in every programmer's library, whether he or she be a professional programmer or a hobbyist.

This column will help illustrate some of the more useful public domain programs. Many interesting programs are actually updated versions of an existing program, in which case the most recent will usually be mentioned and the fat trimmed off.

Each column will take a random walk through the mountain of available material and pull out samples for examination. Sometimes we will examine a specific volume of one of the software libraries, and sometimes we'll look at programs from several different sources.

In most cases, the CP/M operating system will be used if not specified otherwise, although MS-DOS (PC-DOS), UNIX, and more specialized operating systems will not be ignored. All of the software to be reviewed should be readily accessible to all readers through one of the sources I'll cover. Comparisons with available commercial (or other public domain) programs will be used where applicable, but remember that all the comments are those of the author and represent the biased view of one slightly eccentric programmer.

Where is public domain software stored?

There are a few sources available to most programmers. The first is through one of the two big collectors of public domain material: CP/MUG (CP/M User's Group, 1651 Third Ave., New York, N.Y. 10028) and SIG/M (Special Interest Group for Microcomputers, P.O. Box 2085, Clifton, N.J. 07015). Each have over 100 volumes (disks) of material available, generally in several formats. Catalogs are usually inexpensive (approximately $10.00) and list the program contents for a number of volumes. The prospective public domain user is encouraged to write to the two organizations and inquire. Starting with the next column, new releases of both of these sources will be listed with their contents.

The second major source of public domain material is through RCPM (Remote CP/M) systems, which are accessed by modems. RCPM systems are widely available—a recent list had over 100 entries, all available for the cost of the telephone bill. To use RCPM systems, the owner simply calls the system through a modem, using a terminal program, and follows instructions on the screen. The programs required to download (copy) other programs from the RCPM are also on the systems, although configuration for specific hardware may be required in some cases. Most RCPM setups have the most commonly used files on-line and will load specific volumes of CP/MUG and SIG/M material on request.

The third source, which is overlooked by many programmers, is the local computer user groups or special interest subdivisions. Most major cities have a group devoted to computers in general and dedicated groups for the more popular machines and languages. The advantage to dedicated user groups is that machine-dependent software is available already configured, and the more popular public domain programs will be available in the required formats.

Despite repetition, dull games, and cute applications, the really good public domain material all seems to suffer from a common problem: lack of documentation. Authors of programs don't bother to write adequate manuals for newcomers, so there is a great deal of confusion about what a program will do and how. The authors usually know their programs so well that, because most of their functions and routines seem common sense, they assume others will immediately see the clear, clean wisdom in their code design. This intimate knowledge of the program makes writing a suitable manual difficult.

Additionally, writing good documentation can take longer than writing the program. And the writing can vary to such a great deal in style that some authors would be better not to document their material at all. We hope we can shed some light on the documentation problem in this monthly critique.

And with that said, "Lead on, MacDuff . . ."

If I were marooned on a desert island with only one public domain program available, it would have to be FINDBAD. This program helps solve the most dreaded enemy of computer users: the infamous "BDOS ERROR ON X: BAD SECTOR" message. Media problems are unpredictable and always happen at the absolutely worst possible time (in conjunction with a corollary to Murphy's Law). Usually the disk with the problem is relegated to the garbage pile, but with FINDBAD it can be salvaged.

FINDBAD performs a simple diagnostic test on the media in question whether it is a floppy disk (5¼- or 8-in. and probably the new 3-in. by now) or hard disk drive. FINDBAD reads through all the tracks on the disk and attempts to find any bad sectors that could cause problems. The program is entirely nondestructive; it does not write to the disk (only reads it) and therefore will not harm any data stored on the media unless a bad sector is discovered. If there is a bad sector found in the system or directory tracks, FINDBAD will issue a warning message (it varies depending on the version in use). If the problem is with the system track, FINDBAD aborts to the operating system, but it will continue reading the disk if the problem is on the directory tracks.

Any bad sectors discovered are isolated from the other files by being stored in a new file that FINDBAD creates called "[UNUSED].BAD". The FCB (File Control Block) has pointers to all bad sectors on the disk, rendering them immune to further use by the operating system and effectively making the media usable again. FINDBAD will display a count of all bad sectors located and give their number. The program can be run repeatedly and if new bad sectors are located, they will be tagged in to an existing [UNUSED].BAD file if there is one.

FINDBAD is started by typing the name and the drive to be tested, such as "FINDBAD B:". If no drive is specified, the default drive is assumed. When run, FINDBAD will display a sign-on message indicating the version and then issue a series of status reports. These include the tracks being examined, such as "Testing System Track" and "Testing Directory", and will show the progress of the program by printing an asterisk on the console for each sector tested. When it locates a bad sector, FINDBAD retries a few times, then prints a message such as "Bad Block: 0AH".

However, as FINDBAD creates a new file in which to collect the bad sectors, the file must be maintained to keep the disk clean. In other words, issuing an *ERA* *.* command will erase all the files on the disk, including [UNUSED].BAD, which of course leads to bad sector messages again. This is the usual cause of frustration for FINDBAD users.

FINDBAD.COM is available in many versions and on many user disks. The latest versions are all machine independent and will read any kind of disk with no initialization (i.e., you don't have to tell it the format or size of the disk). Hard disk users will find this program indispensable, as a bad sector can totally wreck a

GOOD NEWS!

C for the 6809 WAS NEVER BETTER!

INTROL-C/6809,Version1.5

Introl's highly acclaimed 6809 C compilers and cross-compilers are now more powerful than ever!

We've incorporated a totally new 6809 Relocating Assembler, Linker and Loader. Initializer support has been added, leaving only bitfield-type structure members and doubles lacking from a 100% full K&R implementation. The Runtime Library has been expanded and the Library Manager is even more versatile and convenient to use. Best of all, compiled code is just as compact and fast-executing as ever - and even a bit more so! A compatible macro assembler, as well as source for the full Runtime Library, are available as extra-cost options.

Resident compilers are available under **Uniflex, Flex** and **OS9.**

Cross-compilers are available for **PDP-11/UNIX** and **IBM PC/PC DOS** hosts.

Trademarks:
Intro-C, Introl Corporation
Flex and Uniflex, Technical Systems Consultants
OS9, Microware Systems
PDP-11, Digital Equipment Corp.
UNIX, Bell Laboratories
IBM PC, International Business Machines

For further information, please call or write.

INTROL CORPORATION
647 W. Virginia St.
Milwaukee, WI 53204
(414) 276-2937

CIRCLE 25 ON READER SERVICE CARD

hard disk's use. A readily available source of FINDBAD is on SIG/M Volume 86, which will handle both 5¼- and 8-in. disks. (Versions before FINDBAD 3.8 are for 8-in. disks only, so check the version.) A version for CP/M-86 is available on SIG/M Volume 96. FINDBAD is also available on just about every RCPM in the continent, and some offer the assembly language version for modification, if required.

Updates of FINDBAD are available—some with extra facilities—but some are also destructive. FINDBAD is small and easily used. Stick with it. There are many commercial programs that do exactly what FINDBAD does but cost much more. None seem to offer anything that justifies the extra money.

Because *COMPUTER LANGUAGE* is devoted to the more advanced computer user, a more specialized program or set of programs will be examined each month to round out the column. In this initial offering, an assembler, a disassembler, and a cross assembler will be briefly highlighted. (This topic will be returned to in future columns.)

CP/M's ASM program is used by most of the CP/M world as a standard assembler, but a few enhanced versions available through public domain sources will quickly relegate ASM to the unused disk files. Ward Christensen (whose name will appear in this column on a frequent basis—he is the "dean" of the public domain world) has offered a program called LINKASM, which not only is faster than ASM but offers a few features that Digital Research left out.

LINKASM will allow any number of ASM files to be assembled into one HEX file, acting as a sort of linker (hence the name). A symbol table for use with programs such as SID (Symbolic Instruction Debugger) can be generated during assembly as an option. (However, the symbol table is not completely alphabetically sorted. Only the first letter is used as a sort key so an external sort can be used if an exact listing is required.)

Linking is accomplished by adding the LINK statement to the end of the source code to be linked. The *LINK* command is placed in the opcode field, and the file to be linked is placed in the operand field. The last file in the linking series must end with an END statement to signify completion of linkage. This raises a slight problem: no file to be linked can contain the END statement except as a final command. But this should not raise any serious problems for programmers.

LINKASM is run precisely like ASM in that the command *LINKASM XXXXXXXX.ABC (S:)* is issued. The "XXXXXXXX" is the filename, A is the source drive, B is the destination drive of the HEX listing, and C is the destination drive for the PRN listing. The optional *S:* command indicates the drive on which to place the symbol table.

LINKASM is available as a COM file on CP/MUG Volume 36, and the source code can also be obtained, if required. Z80 assemblers are also available, but most have bugs in them which can lead to fatal errors. For the adventurous, try ASMX on CP/MUG Volume 16. (Note, however, that ASMX has been known to destroy disks if a HEX and PRN destination other than the default drive is specified.) ASMX uses the full set of Zilog mnemonics, but with the ever-impending threat of destroyed disks, it tends to be only for those with disk space to waste. (A full Z80 assembler will be described in an upcoming column.)

For a disassembler, Christensen again delivers. RESOURCE is an 8080 disassembler that is irreplaceable for machine language programmers. (A Z80 version called ZESOURCE or REZ is also available for handling both Zilog and TDL mnemonics.) Christensen has written a very good manual to go with the disassembler, although a couple of readings may be required on first use.

The C Interpreter:
Instant-C™

Programming in C has never been Faster.
Learning C will never be Easier.

***Instant-C*™** is an optimizing interpreter for the C language that can make programming in C three or more times faster than when using old-fashioned compilers and loaders. The interpreter environment makes C as easy to use and learn as Basic. Yet ***Instant-C*™** is 20 to 50 times faster than interpreted Basic. This new interactive development environment gives you:

Instant Editing. The full-screen editor is built into ***Instant-C*** for immediate use. You don't wait for a separate editor program to start up.

Instant Error Correction. You can check syntax in the editor. Each error message is displayed on the screen with the cursor set to the trouble spot, ready for your correction. Errors are reported clearly, by the editor, and only once.

Instant Execution. ***Instant-C*™** uses no assembler or loader. You can execute your program as soon as you finish editing.

Instant Testing. You can immediately execute any C statement or function, set variables, or evaluate expressions. Your results are displayed automatically.

Instant Debugging. Watch execution by single statement stepping. Debugging features are built-in; you don't need to recompile or reload with special options.

Instant Loading. Direct generate .EXE or .CMD files at your request to create stand-alone versions of your programs.

Instant Compatibility. Follows K & R standards. Comprehensive standard library provided, with source code.

Instant Satisfaction. Get more done, faster, with better results. Instant-C is available now, and works under PC-DOS*, MS-DOS*, and CP/M-86*.

Find out how ***Instant-C*™** is changing the way that programming is done. Instant-C is $500. Call or write for more information.

Rational
Systems, Inc.

(617) 653-6194
P.O. Box 480
Natick, Mass. 01760

Trademarks: MS-DOS (Microsoft Corp.), PC-DOS (IBM), CP/M-86 (Digital Research, Inc.), Instant-C (Rational Systems, Inc.)

CIRCLE 46 ON READER SERVICE CARD

The power of RESOURCE should not be underestimated. Source code can be obtained that exactly matches the original. As Christensen points out in his introduction to the manual, commercial software vendors pleaded that he not release his product as it would tend to allow unrestrained ripping off of their products. Luckily, he didn't listen to them.

The use of RESOURCE and ZESOURCE (REZ) is too complicated to explain here, but let it be stated that a finer disassembler has not been seen anywhere for use by an experienced assembly language programmer. RESOURCE is available on CP/MUG Volume 42 with documentation, and ZESOURCE (REZ) is on CP/MUG Volume 64 and SIG/M Volume 10. SIG/M Volume 91 contains another Z80 disassembler called DASM, which handles both Zilog and TDL mnemonics.

Cross assemblers are available for a few conversions that may be of use to a CP/M programmer. To convert 8080 mnemonics to Z80 (Zilog) mnemonics, use a program called XLATE available in SIG/M Volume 91. The program has been run several times and produces neat, clean code, although maximum use of the Z80 features is naturally not obtained. However, XLATE does a very good job of cross assembly.

An 8080 to 68000 cross assembler is available on SIG/M Volume 92 under the name of A68K, but this has not been tested by the author.

Finally, a useful program for assembly language programming is COMBINE, available on CP/MUG Volume 36. COMBINE will concatenate a sequence of source files into one large file and, to save space, remove all comments in the process. On the other end of the spectrum, NOTATE on CP/MUG Volume 78 will allow insertion of comments into source files. It rather cleverly scans each line of code and positions the cursor at the start of the comment field if none exists.

Two cross referencers that may be of interest are XREFPRN on CP/MUG Volume 78 and XREFASM on CP/MUG Volume 6. XREFPRN will send a list of cross-references by address to the CP/M LST: device. XREFASM creates a file with line numbering and a cross-reference index of symbols.

Next month, among other material we'll look at directory programs that replace the *DIR* command with a combined directory sort, as well as the *STAT* command. Disk file managers also will be examined.

I look forward with great enthusiasm to writing In The Public Domain in a way that, I hope, will be both interesting and useful to you. If you have any comments, criticisms, or public domain programs that you think deserve mention in this column please send your feedback in to: In The Public Domain, *COMPUTER LANGUAGE*, 131 Townsend St., San Francisco, Calif. 94107. Or, better yet, leave me a message on the new *COMPUTER LANGUAGE* bulletin board computer: (415) 957-9370. There you will also be able to pick up a copy of most of the programs that will be mentioned in this column.

JOHN D. OWENS ASSOCIATES JOHN D. OWENS ASSOCIATES JOHN D. OWENS

Up To 34% Off!

S-100 EQUIPMENT:

COMPUPRO • LOMAS • IMS • ICM • ADVANCED DIGITAL • SEMIDISK • MACROTECH • SYSTEMS GROUP • ACKERMAN • LEHIGH VALLEY • FUTECH • SCION • DIGITAL GRAPHICS

GRAPHICS PERIPHERALS: HOUSTON INSTRUMENTS • ALPHA MERICS • GTECO • HITACHI

IBM PC AND **XT** AT 10% OFF LIST PRICE!

IBM PC LOOK ALIKE! WYSE 1000 W/80186

MODEMS (300 TO 19.2K BAUD)

TERMINALS • **PRINTERS** • **DISKETTES**

$10 DISCOUNT COUPON with purchase of second edition of our elegant 8 and ½ × 11, fifty five page catalogue replete with detailed product specifications. **SECOND $10 DISCOUNT COUPON** for correct solution to technically oriented, difficult but entertaining crossword puzzle. **THIRD DISCOUNT COUPON** for dust covers. Order catalogue by telephone or mail— $3.95, we pay postage. We accept Master, VISA and American Express. Our catalog is not a cheap 'throw-away'. It contains interesting reading material, is beautifully type-set and printed and bound with high quality materials. We have received lavish praise for this catalog. The purchasing agent at a prestigious medical college rated it 9.5 out of 10.

WE EXPORT: Overseas users call (212) 448 6298 or TWX 710 588 2844

JOHN D. OWENS ASSOCIATES, INC.
12 SCHUBERT STREET
STATEN ISLAND, NEW YORK 10305
(212) 448 6283 (212) 448 6298 (212) 448 2913

Syntax → custom software → Constructs

Get the power of your Z80 and the elegance of direct access to CP/M functions from your high level programs with

SYNLIB
utility library

***SYNLIB** consists of MICROSOFT compatible object code that may be called from any high level language that uses MICROSOFT parameter passing conventions.*

***SYNLIB** gives you extremely powerful array and buffer manipulation using the Z80 LDIR instruction; program access to the CP/M CCP console command line; high speed disk block I/O; a high quality random number generator; HEX to ASCII conversion optimized by special Z80 instructions; program chaining and more.*

And, because our programmer abhors a vacuum, each 8" floppy comes packed with MODEM7 and other valuable public domain software. We've included documentation and available source, so that you too may join the free software movement.

SYNLIB *$50.00*
8" SSSD CP/M format
SOURCE: $100.00
Licensing for commercial use available.

SYNTAX CONSTRUCTS, INC.
14522 Hiram Clarke
Houston, Texas 77045
(713) 434-2098

CP/M is a registered trademark of Digital Research, Inc. Microsoft is a registered trademark of Microsoft Corp. Z80 is a registered trademark of Zilog.

CIRCLE 52 ON READER SERVICE CARD

EXOTIC LANGUAGE OF THE MONTH CLUB

Discovering SNOBOL4

By Ralph E. Griswold and Madge T. Griswold

Some of the more exotic programming languages, like APL and LISP, have sizable followings of ardent devotees. A lesser-known language in the same category is SNOBOL4, a general purpose programming language with powerful string manipulation and pattern-matching facilities.

The original version of SNOBOL was developed at Bell Telephone Laboratories in 1962 by a small group of researchers who were working on symbolic mathematics and theorem proving. SNOBOL quickly became popular because of the almost total lack of string-processing facilities in other programming languages that were available at the time. A repertoire of built-in functions was added to produce SNOBOL2, followed by programmer-defined functions (procedures) in SNOBOL3. SNOBOL4 provided extensive enhancements to the pattern-matching facilities of the earlier SNOBOL languages and also added a number of more sophisticated features.[1,2]

SNOBOL4 is often taught in courses on comparative programming languages and nonnumerical computation. It is used in academic and industrial environments for a wide variety of applications that call for complex string processing, such as linguistic analysis, cryptography, artificial intelligence, molecular genetics, and document formatting. Because SNOBOL4 programs tend to be easy to write, SNOBOL4 also is commonly used for "one-shot" applications, such as analyzing programs and reformatting data files.

SNOBOL4 has been implemented on most large- and medium-scale computers. More recently, there have been several implementations for the IBM PC and its compatibles.

This month's Exotic Language of the Month Club column shows some examples of SNOBOL4's uses and features, gives a few short program examples, and provides pointers to where one can find out more about SNOBOL4, including sources for SNOBOL4 translators.

SNOBOL4 emphasizes ease of programming and run-time flexibility.

Details that the programmer must specify in most other programming languages are handled in SNOBOL4 by the implementation. Here are the main linguistic features of SNOBOL4:

- A variety of data types are supported, including strings, integers, real (floating-point) numbers, patterns, arrays, and tables
- Variables are untyped; there are no type declarations
- Type conversion (coercion) is performed automatically according to context
- Storage management is automatic; there are no storage declarations. Space is allocated as needed, and garbage collection is performed automatically without programmer intervention
- Data objects, such as strings and arrays, can be arbitrarily large and are limited primarily by memory capacity
- Input and output are implicit
- Programmer-defined functions can be called recursively
- Binding times are late; most language operations can be defined or changed during program execution

Because of its early origins, SNOBOL4 lacks the syntactic sophistication and control structures commonly found in more recently developed programming languages. A SNOBOL4 program consists of a sequence of statements that are executed in order. This sequential order can be altered with conditional transfers, which are governed by the success or failure of computations (as opposed to using Boolean values). Some operations always succeed. An example is *SIZE(s)*, which computes the size of the string *s*. Other operations, such as *LT(i,j)*, which tests whether *i* is numerically less than *j*, may succeed or fail depending on whether or not the test succeeds. A conditional goto, which appears at the end of a statement, transfers control depending on the success or failure of the statement. Thus,

```
        LT(i,j)    :S(yes)F(no)
```

transfers control to a statement labeled *yes* if *i* is less than *j* but to a statement labeled *no* otherwise.

Since SNOBOL4 provides no other control structures, the programmer must construct loops. An example is:

```
          i = 0
while     LT(i,10)           :F(done)
          i = i + 1
          new[i] = old[i]    :(while)
```

As indicated, labels appear at the beginning of statements and a goto without an *S* or *F* specifies an unconditional transfer. This program segment simulates a *for* loop, testing and incrementing *i* until the limit of 10 is reached, during which values are copied from one array to another. SNOBOL4 programmers learn to simulate conventional control structures, although it requires discipline to write well-structured SNOBOL4 programs.

Input and output are particularly simple in SNOBOL4. When a value is assigned to the identifier *OUTPUT*, a copy of that value is written out. Similarly, when the value of *INPUT* is used in a computation, a new line is read in and becomes the (string) value of *INPUT*, which is used in the computation. If an end-of-file is encountered when reading, the reference to *INPUT* fails. Thus, the line

```
copy    OUTPUT = INPUT    :S(copy)
```

copies the input file to the output file. This example illustrates how easy it is to program in SNOBOL4—in fact SNOBOL4 is famous for its one-liners.

There are provisions for associating different files with different identifiers for both input and output. Some implementations support I/O format conversions and random-access I/O.

Pattern matching is the heart and soul of SNOBOL4. There is a pattern-matching statment, which has the form

```
        s    p
```

in which *s* is a subject string that is matched according to the pattern *p*. The SNOBOL4 programmer can use many different kinds of patterns to analyze the structure of a string and to break it down into smaller components.

The simplest patterns are positional and refer to locations in strings. Examples are *LEN(n)*, which matches any *n* characters of a string, and *TAB(n)*, which matches up to the *nth* character of a string. Other patterns are lexical and depend on the occurrence of specific characters in a string. Examples are *ANY(s)*, *BREAK(s)*, and *SPAN(s)*, which match any single character in the string *s*, up to any character in *s*, or a sequence of characters in *s*, respectively. For example, *ANY("aeiou")* matches any single vowel.

Patterns can be combined to match in succession by writing them one after another, as in

```
     LEN(10)     BREAK(".,?;:")
```

This pattern first matches 10 characters and then looks for one of the given punctuation marks in the remainder of the subject. Alternatives also can be specified, as in

```
     BREAK(".")       |      LEN(1)
```

in which the vertical bar separates the two alternatives. This pattern tries to locate a period. If that fails, it matches just the first character of the subject.

A variable can be attached to a component of a pattern so that if the pattern matches, the matched substring is assigned to the variable. This is done with the "dot" operator, as in

```
     BREAK(letter)   (SPAN(letter) . wrd)
```

which is a pattern that matches up to a letter, using *BREAK*, and then matches consecutive letters, using *SPAN*. The substring matched by *SPAN* is assigned to *wrd*, effectively picking the first "word" out of the subject.

Patterns are data objects and can be assigned to variables, as in

```
      wpat = BREAK(letter)
  +       (SPAN(letter) . wrd)
```

(A plus symbol at the beginning of a line in SNOBOL4 indicates continuation of the preceeding line.) Thus, a pattern can be constructed once and used in many places in a program. Larger patterns can be constructed from smaller ones so that very complex specifications of string structures can be built up incrementally.

A pattern-matching statement may succeed or fail depending on whether or not the pattern matches. For example, the program

```
read   line = INPUT              :F(done)
       line   ANY("aeiou")       :F(read)
       OUTPUT = line             :(read)
```

writes out only those input lines that contain a vowel.

The subject string can be modified in a replacement statement, which has the form

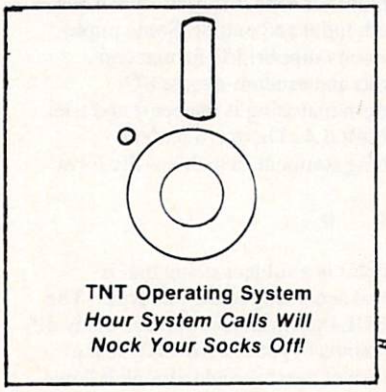
```
     s1      p = s2
```

in which the substring of *s1* that is matched by *p* is replaced by *s2*. For example, using the pattern *wpat* given above, the program

```
read    text = INPUT          :F(done)
findw   text wpat = ""        :F(read)
        OUTPUT = wrd          :(findw)
```

writes out all the words in the input file. In the second statement, if *wpat* matches, the string up to the word and the word itself are deleted and replaced by the zero-length null string. The word that is found is assigned to *wrd* as shown in the previous pattern and written out. When no more words are found, the next line of input is read and processed.

The pattern-matching repertoire of SNOBOL4 is too extensive to describe in detail here. It includes not only simple patterns, such as those given above, but complex ones as well. It is straightforward to write patterns that correspond to context-free grammars (like those specified by production systems such as BNF). For example, the statements



CAPA *(Computer Analyst & Programmers Assoc.)*

THE *Association for the Professional.*

An organization dedicated to the improvement of programming languages, operating systems, system documentation, and I/O devices. The hours you spend finding OEM bugs need not be in vain. Join CAPA and there will be no need for you to rediscover the wheel.

TNT Operating System
Hour System Calls Will Nock Your Socks Off!

Write or Call
for more information:

CAPA
P.O. Box 5762
Rockford, IL 61125-0762
(815) 962-4400

The Force Will be with you.

CIRCLE 6 ON READER SERVICE CARD

DESIGNER SCREENS

"A 100 to 1 Productivity
Increase Over Coding"

Provides full-screen editing of terminal screen design images. And, a linker that generates self-relocating, 8080 machine language, run-time support.

Makes it easy to implement on-screen forms, menus, help screens, boiler-plate notices, and even simple animation.

Run-time support for input includes: data type control, decimal alignment, a type ahead buffer, end-user edit commands, and everybody's favorite, "Fred's Magic Window."

Fred's Magic Window can display field-by-field input instructions as needed, automatically.

Can be used with any computer language that allows programmed calls to CP/M 2.2. Great with assembly language or BDS C.

Runs on 80 × 24 or larger ASCII terminals. Supports five display attributes and line drawing. Designs are transportable between installed terminals.

Manual only:	**$ 10.00**	**(Check it out!)**
Software:	**185.00**	**(Supplied on: 8" SSSD CP/M or call.)**
Complete:	**$195.00**	

(Calif. residents add sales tax)

Austin E. Bryant Consulting
P.O. Box 1382, Lafayette, CA 94549
(415) 945-7911

MasterCard VISA

CP/M is a trade mark of Digital Research
BDS C is a trade mark of BD Software

CIRCLE 5 ON READER SERVICE CARD

```
var = ANY("xyz")
add = ANY("+-")
mul = ANY("*/")
elem = var | ("(" *expr ")")
term = *elem | (*elem mul *term)
expr = *term | (*term add *expr)
```

produce a pattern *expr* that matches simple arithmetic expressions such as *x*, *(x+y)*, *x/(z−y)*, and so on. The stars in front of the references to patterns delay their evaluation until pattern matching takes place, producing the effect of mutual recursion among the patterns.

It is also possible to write patterns that are context-sensitive; in fact, it is possible to perform any computation at all during pattern matching. The pattern-matching process itself, which is built into SNOBOL4, is completely general and provides an exhaustive search and backtrack algorithm. There are also patterns that allow the programmer to limit the scope of backtracking in order to avoid inefficient or unnecessary matching.

While pattern matching is the feature of SNOBOL4 that is most well-known and widely used, SNOBOL4 has a number of other useful features such as tables. A table is like an array, except that a table can be subscripted with any kind of value (such as a string), not just by integer position. A common use of tables is illustrated by the following program segment:

```
         w = TABLE()
read     text = INPUT             :F(sort)
findw    text wpat = ""           :F(read)
         w[wrd] = w[wrd] + 1
+                                 :(findw)
sort     w = SORT(w)
         i = 0
incr     i = i + 1
         OUTPUT = w[i,1] ":" w[i,2]
+                                 :s(incr)
```

The function *TABLE()* creates an empty table, which is assigned to *w*. The input file is read as in the earlier example and is matched with *wpat*. The words are used to subscript *w*, and their count is incremented. (The initial value for a new element in a table is the null string, which is automatically converted to 0 in arithmetic operations.) When the input file is exhausted, *w* is sorted, which produces an *n*-by-2 array, where *n* is the number of words in the table. (Some implementations of SNOBOL4 do not include *SORT*, but this function can be written by the user.) At the end of the program there is a loop through the array that writes out each word and its count with a separating colon. Concatenation is automatic when strings are written in succession, as in the assignment to *OUTPUT*. When the value of *i* exceeds the size of the array, the array reference *w[i,1]* fails (in a fashion similar to failure on the end of an input file).

Note that this program works properly regardless of the number of different words in the input file; the table grows in size automatically as new words are added. Automatic type conversion is performed when the (integer) value of *w[i,2]* is concatenated for output. The use of failure of an out-of-range array subscript is idiomatic in SNOBOL4 and saves a separate test.

SNOBOL4 has a number of esoteric features as well. Operators and built-in functions can be redefined during program execution. The entry points and local identifiers of defined functions can be changed dynamically. It is even possible to create new identifiers during program execution. In fact, strings can be converted into new statements during program execution so that SNOBOL4 programs can modify themselves.

While such esoteric features are rarely needed, they offer maximum flexibility and often can be used to advantage. In truly sophisticated applications, they can

TMS320 Software from COMPUTALKER

All you need is . . . IBM-CP/M or Apple II to develop software for Texas Instruments' TMS320 Digital Signal Processor

TASSM — a TMS320 Assembler

TASSM is currently available for the IBM PC using MS-DOS, CP/M systems using either 8080 or Z-80, and for Apple II using CP/M or Apple DOS. Under CP/M, a 32K system is required.

TASSM Binary License	CP/M-80 or APPLE DOS	$190.00
	MS-DOS for IBM PC	$325.00

S320 — an Interactive TMS320 Software Simulator

S320 is currently available for the IBM PC using MS-DOS, CP/M systems using a Z-80, and for Apple II or II+ using CP/M or DOS 3.2 or 3.3 Under CP/M, a 48K system is required. Under Apple DOS, a language card is required.

S320 Binary License	CP/M-80 or APPLE DOS	$190.00
	MS-DOS for IBM PC	$325.00

Software update support is available at $50 annually.

CP/M is a trademark of Digital Research, Inc.
Apple is a trademark of Apple Computers, Inc.
MS-DOS is a trademark of Microsoft, Inc.

For further information, contact: D. Lloyd Rice

COMPUTALKER
1730 21st St., Santa Monica, CA 90404
(213) 828-6546

CIRCLE 11 ON READER SERVICE CARD

SMALL C FOR IBM-PC

Small-C Compiler Version 2.1 for PC-DOS/MS-DOS
Source Code included for Compiler & Library
New 8086 optimizations
Rich I/O & Standard Library

$40

CBUG SOURCE LEVEL DEBUGGER FOR SMALL C

Break, Trace, and Change variables all on the source level
Source code included

$40

Datalight
11557 8th Ave. N.E.
Seattle, Washington 98125
(206) 367-1803

ASM or MASM is required with compiler.
Include disk size (160k/320k), and DOS version with order.
VISA & MasterCard accepted. Include card no. & expiration date.
Washington state residents include 7.9% sales tax.
IBM-PC & PC-DOS are trademarks of International Business Machines
MS-DOS is a trademark of Microsoft Corporation.

CIRCLE 17 ON READER SERVICE CARD

make it simple to do things that are not feasible in other programming languages. It is not surprising that programming in SNOBOL4 can be fun—even exciting.

As mentioned earlier, SNOBOL4 has been implemented on a wide range of computers. A complete list is given in *Implementations of SNOBOL4*.[3] There are currently three implementations for the IBM PC and compatibles.

Minnesota SNOBOL4 is a full implementation of the standard language. It requires a minimum of 128K bytes of memory (more is recommended) and supports the large memory model. Integers are 32 bit. Floating-point arithmetic is available if a math co-processor is installed.

SNOBOL4+ is a full implementation of the standard language, with some extensions. It requires a minimum of 128K bytes of memory (more is recommended) and supports the large memory model. Integers are 16 bit. Real numbers have 64-bit precision; floating-point emulation is provided if a math co-processor is not installed. Linkage to functions written in assembly language is provided to extend the SNOBOL4 repertoire.

MACRO SPITBOL is a high-performance implementation of a dialect of SNOBOL4 that supports most standard features and offers a number of extensions. It requires 192K bytes of memory and uses the small memory model. Integers are 16 bit. Floating-point arithmetic is not provided. ■

References

1. Griswold, Ralph E., James F. Poage, and Ivan P. Polonsky. 1971. *The SNOBOL4 Programming Language*, second edition. Prentice-Hall Inc., Englewood Cliffs, N.J.
2. Griswold, Ralph E. and Madge T. Griswold. 1973. *A SNOBOL4 Primer*. Prentice-Hall Inc., Englewood Cliffs, N.J.
3. Griswold, Ralph E. *Implementations of SNOBOL4*. 1984. Technical report S4D57, Department of Computer Science, The Univ. of Arizona, Tucson, Ariz.

Ralph Griswold was one of the originators of the SNOBOL languages. He is currently a professor of computer science at the University of Arizona with research interests in programming language design, implementation, and nonnumeric programming.

Madge Griswold has been associated with SNOBOL4 since 1968. She is a free-lance writer and consultant on computing applications and computer-based publication.

Interested in writing, reviewing software, or refereeing manuscripts for

COMPUTER LANGUAGE

For information contact:
Craig LaGrow/Editor
131 Townsend St.
San Francisco, CA 94107
(415) 957-9353
B.B.S.#: (415) 957-9370

eXchanger
CP/M ⟷ ISIS
Emulator

Now move files and programs between your CP/M-80 system and your Intel Series I or II MDS! The ICX package provides complete bidirectional file conversion capability, and even **allows execution** of ISIS-II programs under CP/M using the ISE emulator. The ICX Package is composed of the following two programs:

ICX A Deluxe bidirectional file conversion utility which works with your CP/M system and an 8" floppy drive to provide complete manipulation of an ISIS-II diskette. Takes directories, deletes files, and even initializes a blank disk with the ISIS file structure. Complete C source included. **$89**

ISE An ISIS-II Emulator which allows ISIS programs to run on any CP/M-80 system. Support for all ISIS-II system and monitor calls makes your CP/M micro look like an MDS! Supports banked memory. Complete MAC source included **$89**

Complete ICX Package (ICX & ISE) $175

Supplied on Single Density 8" Disk
CP/M Digital Research, Inc. ISIS-II Intel Corp

Western Wares
Box C
Norwood CO 81423
(303) 327-4898

CIRCLE 55 ON READER SERVICE CARD

DO-IT-YOURSELF CROSS-ASSEMBLERS
LANGUAGE TRANSLATORS
STRUCTURED ASSEMBLY LANGUAGE
$129.95

► The XLT Macro Processor is used as a pre-processor to your favorite assembler. A file of definitions is read by XLT and compared to the input text. A match causes it to be replaced with the definition's contents.
► XLT allows up to ten arguments in a definition, args. with spaces or tabs, increment/decrement strings, a stack, conditional text replacement.
► Includes definitions to translate Z80 to 8080 mnemonics and vice-versa, an 8748 cross-assembler, definitions to implement a C-structured assy. language, allowing ***add a,(hl)*** to be written ***a + = *hl.***

XLT costs $129.95 plus 5.00 P&H. For CP/M 2.2 8" single density disk Z80 systems.

Send check or money order to:

L.A. Software
6708 Melrose
Los Angeles
Calif. 90038
213/932-0817

CA res. add sales tax CP/M is a trademark of Digital Research

CIRCLE 28 ON READER SERVICE CARD

Elegance
Power
Speed

C

C Users' Group
Supporting All C Users
Box 287
Yates Center, KS 66783

CIRCLE 15 ON READER SERVICE CARD

MATH
SUBROUTINE
LIBRARY

Now, a library of Numerical Methods Subroutines for use with your FORTRAN programs.

Over sixty subroutines of:

FUNCTIONS
INTEGRATION
MATRICES
NON-LINEAR SYSTEMS
INTERPOLATION
LINEAR SYSTEMS
POLYNOMIALS
DIFFERENTIAL EQ

Versions available for several FORTRAN compilers running under **CP/M-80** and **MS-DOS (PC-DOS).**

Cost, $250.
Manual available, $25.

microSUB:MATH

CP/M and MS-DOS are trademarks of Digital Research Corp and MicroSoft Corp. respectively.

foehn consulting, PO Box 5123, Klamath Falls, OR 97601 503/884-3023

CIRCLE 20 ON READER SERVICE CARD

EASY
To Use!

Developed in England by Southern Software

SBE

Z80 / 8088 (8086 / 80186 / 8087) machine-code development system. With latest Reduced Instruction-Set philosophy.

☐ SBE/TRS 80 (All DOS) **$100** + $3 s/h
☐ SBE/PC (PC-DOS/MS-DOS **$160** + $3 s/h

Allen Gelder Software
(415) 681-9371
Box 11721 San Francisco, CA 94101

CIRCLE 22 ON READER SERVICE CARD

COMPUTERVISIONS

An Interview with Charles Moore—founder of Forth

By Leo Brodie

Of the dozen most popular computer languages of our time, Forth is certainly one of the most unusual. It's not surprising that its creator, Charles H. Moore, is a self-proclaimed maverick who admits, "While everyone is marching along one path, I'm bound and determined to find a different one."

Back in the late sixties Moore had no idea he was starting a movement. He was merely a programmer who, in the process of working on applications, developed a set of tools to make his work easier. At no time, he claims, did he actually sit down to write a programming language. Instead, he kept extending and changing his personalized programming environment until 1971 when, at the National Radio Astronomy Observatory, he developed what he calls the first use of modern Forth.

Many of the concepts Moore initially promoted—amid general disinterest or disagreement—are now widely accepted. For instance, the use of very small routines that can be put together as needed (rather than monolithic immutable programs) seems to be the cornerstone of the C philosophy, as it is of Forth's. In fact, Forth is designed to minimize the expense of subroutine calls to such an extent that they aren't even treated as subroutines but simply as "words." Forth programming consists of defining words in terms of other words.

Moore is delighted that people in the mainstream of the computer industry are acknowledging what he feels is the right direction. But he admits to being a little irked that the establishment hasn't paid any attention to Forth.

"We've had more of value to say than I think they've given us credit for," claims Moore. "We aren't doing anything they've advised against, but we have done things in an unfamiliar format."

Although proud of Forth's success, Moore admits that he has failed to popularize it. Others have succeeded more than he— including FORTH Inc. and the Forth Interest Group (FIG). For years, FIG has been giving away listings of Forth systems for no more than the copying cost. It's safe to guess that there are more FIG-based systems in use today than any other variety.

Like many in the past who have made landmark discoveries in the arts and sciences, Moore has received little appreciation for his contributions, and he has been undercompensated for them as well. Although specific implementations may be proprietary, Forth itself is in the public domain, and Moore receives no royalties. Far from being bitter, Moore recognizes that the immediate availability of Forth systems has been good for its popularity.

Since Forth was first developed, Moore estimates that he has written 10 times as many programs as he could have without it. But in the ensuing decade, while working as a contract programmer both for FORTH Inc. and now for his own company, Charles H. Moore & Associates, he became increasingly frustrated with the limitations of available hardware.

In 1980 he addressed FIG's national convention with the comment, "Hardware today is in the same shape as software was 20 years ago. There is no point in trying to optimize software any further until we have taken the first crack at optimizing the hardware."

Not one to quietly accept anyone's "givens," Moore now stands on the threshold of his second major achievement: the Forth chip. Since Forth is a virtual machine, existing versions of Forth are actually emulations built over the architecture of the particular processor. But Forth's simplicity makes it a perfect candidate to be cast in silicon as its own architecture.

After several years of waiting, Moore has finally received sufficient funding to produce the chip. By the end of this summer, a prototype chip that will make Forth programs significantly faster should be available.

Photo Credit: Marlin Ouverson, *Forth Dimensions*

"It's a hummer," says Moore. "It has a stunning instruction set, with nearly all the Forth instructions executing in one cycle. It has lots of I/O (120 pins) and all the programmable versatility we've been hoping for."

The chip will be produced by a company called Novix Inc., located in Los Gatos, Calif.

In retrospect, Moore regrets not getting involved in hardware sooner. He admits, "I was somewhat expecting that someone else would build the Forth computer. What it needed was commitment. It's taken an enormous emotional effort to build this machine—more so than the technical effort required."

As soon as he finishes working on the chip, Moore expects to hang out his shingle as a contract hardware/software designer for "sufficiently interesting projects." Particularly, he'd like to design tools that make hardware design more productive. "I can do software in an afternoon. Hardware takes me a week to design. I'd like to be able to design hardware as easily as I can design software. The chip will help," says Moore.

The common thread to all of Moore's work, including his most advanced applications, is simplicity. "All too often the problem is in the details. Too many details prevent you from seeing a different approach," he says.

FOR TRS-80 MODELS 1, 3 & 4
IBM PC, XT, AND COMPAQ

The MMSFORTH System. Compare.

- The speed, compactness and extensibility of the MMSFORTH total software environment, optimized for the popular IBM PC and TRS-80 Models 1, 3 and 4.
- An integrated system of sophisticated application programs: word processing, database management, communications, general ledger and more, all with powerful capabilities, surprising speed and ease of use.
- With source code, for custom modifications by you or MMS.
- The famous MMS support, including detailed manuals and examples, telephone tips, additional programs and inexpensive program updates, User Groups worldwide, the MMSFORTH Newsletter, Forth-related books, workshops and professional consulting.

MMSFORTH

A World of Difference!

- Personal licensing for TRS-80: $129.95 for MMSFORTH, or "3+4TH" User System with FORTHWRITE, DATAHANDLER and FORTHCOM for $399.95.
- Personal licensing for IBM PC: $249.95 for MMSFORTH, or enhanced "3+4TH" User System with FORTHWRITE, DATAHANDLER-PLUS and FORTHCOM for $549.95.
- Corporate Site License Extensions from $1,000.

If you recognize the difference and want to profit from it, ask us or your dealer about the world of MMSFORTH.

MILLER MICROCOMPUTER SERVICES
61 Lake Shore Road, Natick, MA 01760
(617) 653-6136

Forth is rigorously simple; many would think even Spartan. This is the way Moore prefers it. The more you can take out of a system, he feels, the better off you are in usability, reliability, ease of learning, and performance. The key, of course, is the extensibility that Forth provides.

Moore thinks of himself as more of a computer engineer than a computer scientist. "I've pondered the problems of artificial intelligence, heuristic programming, etc. But I'm not one to do a theoretical study on something with a low probability of success. I prefer to solve the immediate problem."

"If we invent machines that act intelligently, they won't come out of the artificial intelligence community—they'll come from someone hacking around in the garage. Nobody knows how to do it; it's as likely to be stumbled upon by accident as by serious research effort."

"I don't even think artificial intelligence is a worthwhile goal," he continues. "Machines should do machine-like things, not act like human beings. An automobile with a sense of self-preservation—one which merely won't damage itself as far as it's able to prevent—that would be an intelligent machine and a worthy goal. It doesn't need to hold a conversation with you."

Moore is married to a warm and witty woman named Min, who possesses a special talent for weaving. Together with their teenage son, Eric, they have lived in a beach-front house in southern California's south bay for over 10 years. But this summer the Moores, seeking more quiet and solitude, plan to move to the redwood-topped mountains overlooking Silicon Valley.

One of Moore's yet-to-be-realized dreams is to establish a Forth university. It would be a remote place where people could practice developing applications in comfort and solitude— solving real problems in short-term projects.

Whether Forth ever becomes as popular as the C programming language remains to be seen. But many thousands of programmers around the world already feel that Moore's unique way of looking at software problems has influenced them for the better.

Leo Brodie is a former employee of Moore's first company, FORTH Inc. He is the author of Starting FORTH. *His* Thinking FORTH, *due out this September from Prentice-Hall, includes numerous interviews with Charles Moore.*

Lower Price!

Eco-C
PROGRAMMING GUIDE
E$

We make C easy . . .

. . . and work!

Whether you're a seasoned pro or just getting started with C, our Eco-C C compiler has everything you need.

- A full C compiler, including long, float and double.
- A library of more than 100 functions for faster program development.
- The compiler generates assembler output (Zilog mnemonics) for processing with Microsoft's MACRO 80 assembler and linker, both of which are included in the price.
- Extremely efficient code (e.g., Knuth's "seive" executes in 15.8 seconds).
- For a limited time, we include a copy of the **C Programming Guide.** The **Guide** and the Eco-C compiler provide an excellent environment for learning C.

Perhaps the best news is that we've lowered the price of Eco-C to $250.00 and it includes a user's manual, the **Guide** and MACRO 80. Eco-C requires a Z80 CPU and CP/M (an 8088 version in the 2nd quarter). To order, call or write:

ECOSOFT INC.
6413 N. College Ave.
Indianapolis, IN 46220
(317) 255-6476

VISA MasterCard

Trademarks: Eco-C (Ecosoft), MACRO 80 (Microsoft), CP/M (Digital Research), Z80 (Zilog)

CIRCLE 34 ON READER SERVICE CARD

CIRCLE 18 ON READER SERVICE CARD

Modula-2/86

Hardware Requirements: IBM PC and compatibles with 256K minimum
Price: $495.00
Available from: Logitech Inc., 805 Veterans Blvd., Redwood City, Calif. 94063. (415) 365-9852
Support: 30-day media warranty, product updates at a nominal charge, technical assistance by phone or mail for registered users.

Logitech's version of Modula-2 is based on the work of Niklaus Wirth, the creator of Pascal and Modula-2. It reflects Wirth's concept of a true business and scientific language, contrasting with Pascal, which was only intended to be a teaching language. Modula-2 is a modern language and has the elements necessary for designing complete systems.

Modula-2 uses the strengths of Pascal and eliminates many of its weaknesses. The standard language has enough strong features that the creation of incompatible dialects should be unnecessary. Machine-specific, low-level subroutines may be implemented in a manner that maintains the modularity of the system. Modula-2 allows the use of true modular programming techniques with strong type checking. Flexibility is provided by data transfer routines that handle different variable types, interrupt handling situations, and hardware/operating system access.

Modula-2/86 is Logitech's full, standard implementation of Modula-2 on 8088- and 8086-based microprocessors. Some of the features provided are:

- Extensive library of standard modules
- 8087 support
- Support of a full 1-megabyte address space of the 8088/8086
- Access to underlying hardware and operating system functions
- Support for the creation of overlays on very large systems
- A symbolic debugger
- Generation of ROM-able code.

The Modula-2/86 system requires an IBM PC or compatible with 192K or more of RAM; 2 double-sided disk drives (300K+ each); and PC-DOS (1.1 or 2.0), MS-DOS, or the CP/M-86 operating system. It is also supported on other configurations of the 8086/8088 running MS-DOS, CP/M-86, Concurrent CP/M-86 or MP/M-86, with enough disk space to hold the compiler and other programs (600K minimum). A printer is not required but is strongly recommended. Those developing software will find that a hard disk is very useful in terms of both convenience and speed of operation.

Compiled Modula-2/86 programs can be executed on any 8088 or 8086 CPU assuming that the system has sufficient memory to hold both the program to be executed and its data. No references to a particular operating system are produced by the compiler. The 8087 numeric data

```
                        Program module
                        (somename.MOD)
                              I
                              I
                       ----------------
Symbol file(s)         I              I
(somename.SYM)    ---> I  Compiler    I
for libraries used     I              I
                       ----------------
                              I
       Link (somename.LNK and Ref file (somename.REF) files
                              I
                       ----------------
Link file(s)           I              I
(somename.LNK)    ---> I   Linker     I
for libraries used     I              I
                       ----------------
                              I
                   Load file (somename.LOD)
                              I
                       ----------------
                       I              I
                       I     m2       I ----> user output
                       I              I
                       ----------------
                              I
                 Memory dump file (MEMORY.PMD)
              created when run-time errors occur
                              I
                       ----------------
Ref file(s)            I              I
(somename.REF)    ---> I   debugger   I ---> screen display
of programs and        I              I
libraries used         ----------------
```

Figure 1.

processor is supported but not required. No software emulation is currently provided for the 8087; in order to develop code for the 8087 you must have one installed.

This review was done using the PC-DOS version of Modula-2/86, release 1.0 on a 2-disk-drive IBM PC with 256K RAM. Setup for use is similar to most PC-DOS software: you must make backups of the three distribution diskettes and prepare working copies with the operating system and certain Modula-2/86 files. Sample programs are supplied in source code form on the distribution diskettes to get started compiling quickly. A brief flow chart on how the package is set up is shown in Figure 1.

The first step is to compose your source code with a text editor of your choice, as Logitech's package does not provide one. (See Listings 1 and 2 for examples of Modula-2 code on the *COMPUTER LANGUAGE* bulletin board service under the file names MODULA1.LTG and MODULA2.LTG. The Modula-2 program in Listing 2 was converted to BASIC in order for you to compare it to the original. See Listing 3 for this BASIC version under file name MODULA3.LTG on the bulletin board service.)

The m2 in the following example is the run-time support program for Modula-2. It is an assembly language program that provides the services for your Modula-2 program to run. M2.EXE is the only stand-alone file in the Modula-2/86 system. Thus it is part of the execution of all other Modula-2 programs as an interpreter-like system. When you have completed your source code, you next invoke the Modula-2/86 compiler by the following keystrokes:

```
m2 b:comp<CR>
```

The compiler then asks for the source file and will supply the default extension of .MOD if no extension is given.

You will then give the source file name, as in this example,

```
source file> exampl<CR>.MOD
  p1
    Terminal in file: B:Terminal.SYM
  p2
  p3
  p4
    termination
```

The setting of the options was:

```
    emulator   (E): off
    stacktest  (S): on
    rangetest  (R): on
    indextest  (T): on
    Codesize:  90 bytes
    Datasize:  1 byte
    (No code for the 8087
        processor was generated)
end compilation
```

The pl . . . p4 indicates the progression through the four passes of the compiler. The compiled files have been written to the disk in drive A and you have been returned to the DOS prompt. The successful compilation will have created the files EXAMPl.LNK (the object module) and EXAMPl.REF (debugger information).

After successful compilation, the program must be linked with the necessary library files needed to perform its given tasks (Figure 2). The Link/Debug disk must be inserted in drive B in the place of the compiler disk. Then the following command is entered:

```
m2 b:link<CR>
```

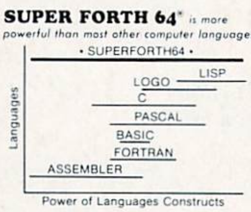
SUPER FORTH 64®

TOTAL CONTROL OVER YOUR COMMODORE-64™
USING ONLY *WORDS*
MAKING PROGRAMMING FAST, FUN AND EASY!
MORE THAN JUST A LANGUAGE...
A complete, fully-integrated program development system.
Home Use, Fast Games, Graphics, Data Acquisition, Business
Real Time Process Control, Communications, Robotics, Scientific, Artificial Intelligence

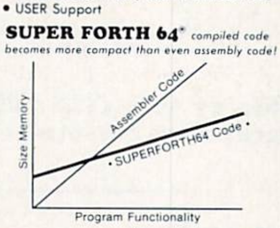
A Powerful Superset of MVPFORTH/FORTH 79 + Ext. for the beginner or professional

- 20 to 600 x faster than Basic
- 1/4 x the programming time
- Easy full control of all sound, hi res. graphics, color, sprite, plotting line & circle
- Controllable SPLIT-SCREEN Display
- Includes interactive interpreter & compiler
- Forth virtual memory
- Full cursor Screen Editor
- Provision for application program distribution without licensing
- FORTH equivalent Kernal Routines
- Conditional Macro Assembler
- Meets all Forth 79 standards+
- Source screens provided
- Compatible with the book "Starting Forth" by Leo Brodie
- Access to all I/O ports RS232, IEEE, including memory & interrupts
- ROMABLE code generator
- MUSIC-EDITOR
- SPRITE-EDITOR
- Access all C-64 peripherals including 4040 drive
- Single disk drive backup utility
- Disk & Cassette based. Disk included
- Full disk usage — 680 Sectors
- Supports all Commodore file types and Forth Virtual disk
- Access to 20K RAM underneath ROM areas
- Vectored kernal words
- TRACE facility
- DECOMPILER facility
- Full String Handling
- ASCII error messages
- FLOATING POINT MATH SIN/COS & SQRT
- Conversational user defined Commands
- Tutorial examples provided, in extensive manual
- INTERRUPT routines provide easy control of hardware timers, alarms and devices
- USER Support

SUPER FORTH 64® *is more powerful than most other computer languages!*



SUPER FORTH 64® *compiled code becomes more compact than even assembly code!*



A SUPERIOR PRODUCT
in every way! At a low
price of only
$96

Call:
(415) 651-3160
PARSEC RESEARCH
Drawer 1776, Fremont, CA 94538

Take this ad to your local dealer, or B. Dalton Bookstore. Phone orders also accepted. Immediate delivery! Dealer inquiries invited. CA residents must include tax.

or C.O.D.

® PARSEC RESEARCH (Established 1976)
Commodore 64 & VIC-20 TM of Commodore

CIRCLE 37 ON READER SERVICE CARD

Q/C

For only $95, Q/C is a ready-to-use C compiler for CP/M. You get complete source code for the compiler and over 75 library functions. Q/C is upward compatible with UNIX Version 7 C, but doesn't support long integers, float, parameterized #defines, and bit fields.

- Full source code for compiler and library.
- No license fees for object code.
- Z80 version takes advantage of Z80 instructions.
- Excellent support for assembly language and ROMs.
- Q/C is standard. Good portability to UNIX.

Version 3.2 of Q/C has many new features: structure initialization, faster runtime routines, faster compilation, and improved ROM support. Yes, Q/C has casts, typedef, sizeof, and function typing. The *Q/C User's Manual* is available for $20 (applies toward purchase). VISA and MasterCard welcome.

THE CODE WORKS

5266 Hollister
Suite 224
Santa Barbara, CA 93111
(805) 683-1585

Q/C, CP/M, Z80, and UNIX are trademarks of Quality Computer Systems, Digital Research, Zilog, Inc., and Bell Laboratories respectively.

CIRCLE 9 ON READER SERVICE CARD

```
Modula-2/86 Linker V1.0 - (c) 1983 Logitech

Output file name: B:exampl.LOD

++ Program map (7 modules included in this layer):
   Modules are listed in order of module code execution.

+ MOD= System        KEY= A78702280DFC  FILE= A:System.LNK
                     CODE= 008A  DATA= 00E3
                     PROC-TABLE= 0002
+ MOD= Keyboard      KEY= A787026FE9FC  FILE= A:Keyboard.LNK
                     CODE= 006D  DATA= 00DA
                     PROC-TABLE= 0002
+ MOD= Display       KEY= A6ED00811F68  FILE= A:Display.LNK
                     CODE= 0065  DATA= 00DA
                     PROC-TABLE= 0002
+ MOD= Termbase      KEY= A787024FACBC  FILE= A:Termbase.LNK
                     CODE= 002A  DATA= 00D3
                     PROC-TABLE= 0002
+ MOD= ASCII         KEY= A6ED007F8F20  FILE= A:ASCII.LNK
                     CODE= 0028  DATA= 00D3
                     PROC-TABLE= 0002
+ MOD= Terminal      KEY= A02101346662  FILE= A:Terminal.LNK
                     CODE= 0006  DATA= 00D2
                     PROC-TABLE= 0002
+ MOD= Exampl        KEY= A8AD0438E362  FILE= B:exampl.LNK
                     CODE= 0000  DATA= 00D1
                     PROC-TABLE= 0022

++ Base (0 modules assumed to be in base layers):

Length of code (in paragraphs): 00D1
Length of data (in paragraphs): 0022
```

Figure 2.

The linker then asks for the file name of the compiled program. The linker will supply the default extension of .LNK if no extension is given. The linker's output is,

```
master file > exampl <CR>.LNK
```

which is linked with:

```
    Terminal  in file: B:Terminal.LNK
    Termbase  in file: B:Termbase.LNK
    System    in file: B:System.LNK
    Keyboard  in file: B:Keyboard.LNK
    ASCII     in file: B:ASCII.LNK
    Display   in file: B:Display.LNK
  name of output file: A:exampl.LOD
  name of map file: A:exampl.MAP
end linkage
```

The linked program is now written to the disk in drive A and can be executed by the Modula-2 system. The command is:

```
m2 exampl <CR>
```

and the following output will appear on the screen:

```
The program worked! (Hit a key)
```

The many other functions of Modula-2/86 can be used to handle process control

LOWER
PROGRAMMING MAINTENANCE
AND DEVELOPMENT COSTS

{SET:**SCIL**™}

The Source Code Interactive Librarian
for microcomputers.

- **SCIL** keeps a historical record of all changes made to the library.
- **SCIL** maintains any source code regardless of language, including user documentation and text material.
- **SCIL** allows software engineers to work with source code as they do now, *using any ASCII text editor.*
- **SCIL** saves disk space by storing only the changes made to the program.
- **SCIL** provides a labeling capability for ease of maintaining multiple versions and multiple releases.
- **SCIL** offers unlimited description in the program library directory.
- *High visibility displays* with varied intensity for ease of viewing insertions and deletions.
- **SCIL** is available on CP/M, MP/MII, MS-DOS, PC-DOS and TurboDOS.

{SET} Get {SET} for Success
{SET:**SCIL**™} is a product of System Engineering Tools, Inc.
645 Arroyo Drive, San Diego, CA 92103

Registered Trademarks: CP/M, MP/M II - Digital Research Inc., MS-DOS, Microsoft Corp., PC-DOS, IBM Corp., TurboDOS, Software 2000, Inc.

For more information call (619) 692-9464.

CIRCLE 53 ON READER SERVICE CARD

'C' COMPILER

AN OUTSTANDING VALUE

"We bought and evaluated over $1500.00 worth of 'C' compilers . . . C/80 is the one we use."
Dr. Bruce E. Wampler, Aspen Software
author of "Grammatik"™

C/80 Full featured C Compiler for CP/M* with I/O redirection, command expansion, execution trace and profile, initializers, Macro-80 compatibility, ROMable code. 49⁹⁵

C/80 FLOATS & LONGS Adds 32 bit data types to C/80 3.0 compiler. Includes I/O and transcendental function library. 29⁹⁵

FREE CATALOG Call or write for 16 page booklet detailing our programming languages LISP/80, RATFOR, Assemblers, and 25 other CP/M products.

The Software Toolworks

15233 Ventura Blvd., #1118
Sherman Oaks, CA 91403

(213) 986-4885
Dealer inquiries invited.

*CP/M is a registered trademark of Digital Research Inc.

CIRCLE 51 ON READER SERVICE CARD

as well as other programming tasks that require concurrency. Modula-2/86 also includes features to set the compiler options, reconfigure the operating system interface, and implement a large number of DOS-standard interrupts. The standard overlay scheme frees the programmer from concerns about the location of an overlay. The relocatable loader handles the necessary functions for the programmer and the overlay may be as large as the physical address space (e.g., a maximum of 1 megabyte). The run-time support provides for language support and additional configuration-dependent functions. These functions include features such as bootstrapping the resident software, setting the memory configuration, dumping memory to disk, etc.

The compiler issues error messages during the compilation process and displays the source code with an indicator at the error location. When a run-time error occurs, the run-time system creates a post-mortem memory dump of the program status (MEMORY.PMD). The symbolic debugger is used to examine this file and to view the errors.

The debugger provides four ways to view the program's error conditions: the data window on variables and parameters, text window on the text of the current module and procedure, process window on the procedure call chain with the address of the call to the next procedure in the chain, and the memory window on the contents of memory around the current address.

The installation of the Modula-2/86 system is greatly expedited by the inclusion of several batch files on the distribution disks. There is a batch file for installation on a floppy-disk system and one for installation on a hard disk system, complete with the commands to create the subdirectories that are useful to separate the various files into logical groups. Also provided is a built-in default name and default search strategy that will automatically check the appropriate paths for a file. Modula-2/86, however, is not copy protected.

The Modula-2/86 system by Logitech is an extremely useful program development environment for the Modula-2 or Pascal programmer. A sufficient number of features are provided to enable the serious developer to handle almost any task that arises. The speed of the compiled programs is excellent and the unavoidable delays of a compiler process can easily be minimized by the use of a hard disk and electronic RAM disk techniques. The missing support of the software emulation for the 8087 is intended to be released soon by Logitech.

The possibilities offered by the Modula-2 environment are as varied as your imagination. Logitech has done an excellent job.

By Chris Jacobs

Professional BASIC

Hardware Requirements: PC-DOS v2.0 or later, minimum 256K memory (384K recommended), 8087 math co-processor chip.
Price: $345.00
Available from: Morgan Computing Co., 10400 N. Central Expressway, Suite 210, Dallas, Texas 75231. (214) 739-5895.
Support: $20.00 for each newest version (includes new disk and updated pages).

There once was a young man who was born blind.

Living in the Middle East 20 centuries ago, he had to accept his reality and do the best he could, which was not much. One day, a famous man from Galilee happened to be passing by him and decided to do something about his blindness. Our friend was able to see!

There is a strong analogy between the blindman in this biblical story and programmers debugging BASIC programs.

Even though BASIC is interactive, we are blind to what is going on in the background. One remedy is to add print statements to show more intermediate results. Another alternative is to use software that patches to a BASIC interpreter and provides some tracing capability.

We just had to accept the fact that these techniques were the best that could be used until Neil Bennet of Morgan Computing gave us Professional BASIC.

Rather than redefining a BASIC interpreter from ground zero, Professional BASIC takes IBM PC BASIC syntax and improves it. The manual assumes the reader is familiar with PC-BASIC and concentrates on what makes the product new and different from PC-BASIC.

Probably the most immediately noticeable difference is the debugging and tracing screens (windows) that allow the programmer to follow, step by step, the events taking place.

The first set of new features relate to the interpreter's use of available system resources. All the available memory can be accessed and used. This overcomes the frustrating 64K segment limitations. Now a programmer can manipulate and process large matrices—as in civil engineering truss designs—and handle more realistic problems.

The use of the 8087 chip by the interpreter further enhances the speed of mathematical calculations. A second boost for the speed comes from the fact that the BASIC code is semicompiled into pseudocode or tokens. This makes Professional BASIC fast.

Another new feature is the dynamic syntax checking which causes every line typed or read from a file to be examined for syntax errors, which are pointed out immediately. Dynamic syntax checking cuts down run-time errors due to syntax, especially in those portions of code (e.g., subroutines) that are rarely executed. It is worthwhile to point out that this feature is not intended for the novice. It is also implemented on Hewlett-Packard machines running a BASIC interpreter. While on the subject of program lines, it should also be mentioned that Professional BASIC allows alphanumeric labels to be used in addition to line numbers in branching and subroutines.

Professional BASIC introduces some new commands. Some enhance the text/variable searching capability. The *FIND* command will search and display those program lines containing a sought variable or label. Program lines containing a user-specified set of characters can be displayed using the *SEARCH* command.

There is another version for these two commands: *FINDL* and *SEARCHL*. They will scan for the sought items and highlight them wherever they occur in the listing.

Professional BASIC has two commands that will furnish the user with a sorted list of variables and labels. However, no accompanying line number is given as is the case with other cross-referencing programs. Another command is the *SRUN* that triggers the tracing mode and windows while running the program.

For those interested in tracing the evaluation of mathematical expressions, Professional BASIC has the *FINTRACE* command. It will display the evaluation step by step so the user can see where things are going wrong. Professional BASIC stresses logically pairing *FOR/NEXT* as well as *WHILE/WEND* statements and lures the user into using more organized loops. As a reward, the *EXITFOR* and *EXITWHILE* commands are presented to exit these loops without using references to line numbers or labels.

Professional BASIC has also modified some PC-BASIC commands. As previously mentioned, the *NEXT* statement must be followed by only one loop counter. No multiple loops can be attached to a *NEXT* statement or vice versa. Concerning loading and saving files, Professional BASIC uses a default name for the last file loaded or saved when none is supplied with the *LOAD* or *SAVE* commands. This will prevent the user from saving an updated program in a misspelled filename or, worse yet, in a similar existing filename (overwriting the latter's contents).

The current version of Professional BASIC has not implemented PC-BASIC related to the following:

- Graphics and sound
- Communication via the RS-232 ports
- Program chaining
- Callable machine language subroutines.

Professional BASIC has enhanced the integer type and the variable names. The integer type variables can have values ranging between plus or minus 2 billion and occupying four bytes of memory each. All arrays must be dimensioned (no-default array size is assumed) with the ability of having a lower range index other than zero or unity. Thus it is possible to declare an array "Year" as:

```
100 DIM Year(1981 to 2000)
```

```
100 ' TEST PROGRAM : AREA UNDER CURVE USING SIMPSON'S RULE
110 DEFDBL A,S,X
120 DIM X(3000)
130 TIME$="0"
140 FOR I = 0 TO 3000
150       X(I) = LOG(SQR(I + .1))
160 NEXT I
170 SUM.ODD = 0
180 SUM.EVEN = -X(3000)
190 FOR I = 1 TO 3000 STEP 2
200       SUM.ODD = SUM.ODD + X(I)
210       SUM.EVEN = SUM.EVEN + X(I)
220 NEXT I
230 AREA = (1/3) * (X(0) + 4*SUM.ODD + 2*SUM.EVEN)
240 LPRINT "TIME = ";TIME$ : LPRINT
250 LPRINT "AREA = ";AREA
260 READ FIRST%,LAST%
270 DATA 1,10 ' Data statment
280 OPEN "O",1,"DATA.DAT"
290 GOSUB STORE.IT ' gosub using a label
300 CLOSE#1
305 END
310 STORE.IT; ' subroutine to save data.
320 FOR I= FIRST% TO LAST%
330      PRINT#1,X(I)
340 NEXT I
350 RETURN
```

Listing 1.

This defines a 20-member array with the lower bound being less of an abstract value. The size of the array is no longer limited to 32,767. Instead it can be as large as two billion.

Professional BASIC allows long variable names for program readability. It has solved the problem involved with lengthy names—namely, spelling errors—and the frustration involved in retyping long names. The programmer simply types the minimum number of characters sufficient to uniquely identify the variable's name, followed by "@." The interpreter will either continue the rest of the name or signal that more characters are needed to identify the sought name.

The manual states that Professional BASIC File I/O is able to access up to 4 billion records. The number of bytes per record can be as high as 65,535, and the number of bytes per file can be as high as 4 billion. This makes Professional BASIC an excellent choice for writing customized data bases on hard disks. However, there are some implementation restrictions with the current version.

The tracing windows in Professional BASIC are, in my opinion, what crown the product. I will use a small program (Listing 1) to show the number of windows available; the program performs numerical integration using Simpson's rule. I also added some commands to show the READ/DATA, GOSUB, and I/O windows.

As the listing shows, there are two *FOR/NEXT* loops that repeat similar calculations 3,000 times. PC-BASIC took 77 sec to execute lines 140 through 230. Professional BASIC took 21 sec (266 percent faster).

I also ran a test by compiling the program with an 8087 BASIC compiler. It took 4 sec (425 percent faster than Professional BASIC) to obtain the result.

While tracing a program, the user has many choices of windows. List-trace windows display the program lines so that each statement is displayed on a separate screen line, properly indented. During a trace run, the executed statement is displayed in reverse video. The user has the

LATTICE®
C Compilers

"My personal preferences are Lattice C in the top category for its quick compile and execution times, small incremental code, best documentation and consistent reliability; . . ."
BYTE AUG. 1983
R. Phraner

". . . programs are compiled faster by the Lattice C compiler, and it produces programs that run faster than any other C compiler available for PC-DOS."
PC MAGAZINE JULY 1983
H. Hinsch

". . . Microsoft chose Lattice C both because of the quality of code generated and because Lattice C was designed to work with Microsoft's LINK program."
PC MAGAZINE OCT. 1983
D. Clapp

"Lattice is both the most comprehensive and the best documented of the compilers. In general it performed best in the benchmark tests."
PERSONAL COMPUTER AGE NOV 1983
F. Wilson

"This C compiler produces good tight-running programs and provides a sound practical alternative to Pascal."
SOFTALK AUG 1983
P. Norton

". . . the Lattice compiler is a sophisticated, high-performance package that appears to be well-suited for development of major application programs."
BYTE AUG 1983
Houston, Brodrick, Kent

To order, or for further information
on the LATTICE family of compilers, call or write:

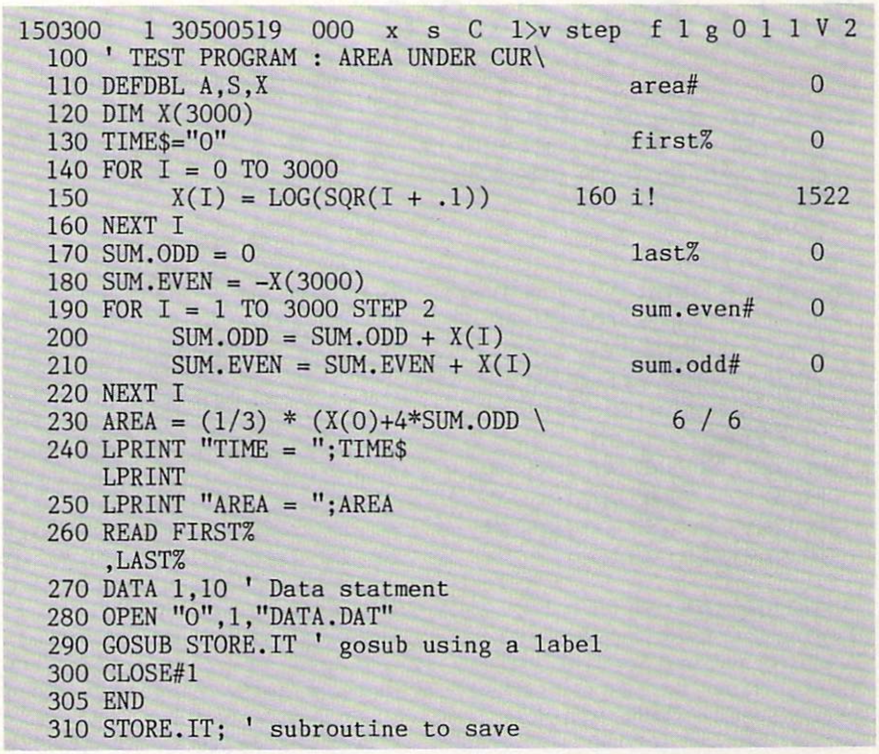
VISA
LATTICE, INC.
P.O. Box 3072
Glen Ellyn, IL 60138
MasterCard
(312) 858-7950 TWX 910-291-2190

CIRCLE 29 ON READER SERVICE CARD

ADVERTISE
in the
October
issue
of
COMPUTER
LANGUAGE

Reservation Deadline:
August 6th

Contact:
Carl Landau
Computer Language
131 Townsend Street
San Francisco, CA 94107
(415) 957-9353



```
150300   1 30500519  000  x  s  C  1>v step  f 1 g 0 1 1 V 2
  100 ' TEST PROGRAM : AREA UNDER CUR\
  110 DEFDBL A,S,X                               area#        0
  120 DIM X(3000)
  130 TIME$="0"                                  first%       0
  140 FOR I = 0 TO 3000
  150      X(I) = LOG(SQR(I + .1))          160 i!         1522
  160 NEXT I
  170 SUM.ODD = 0                                last%        0
  180 SUM.EVEN = -X(3000)
  190 FOR I = 1 TO 3000 STEP 2                   sum.even#    0
  200      SUM.ODD = SUM.ODD + X(I)
  210      SUM.EVEN = SUM.EVEN + X(I)            sum.odd#     0
  220 NEXT I
  230 AREA = (1/3) * (X(0)+4*SUM.ODD \             6 / 6
  240 LPRINT "TIME = ";TIME$
      LPRINT
  250 LPRINT "AREA = ";AREA
  260 READ FIRST%
      ,LAST%
  270 DATA 1,10 ' Data statment
  280 OPEN "O",1,"DATA.DAT"
  290 GOSUB STORE.IT ' gosub using a label
  300 CLOSE#1
  305 END
  310 STORE.IT; ' subroutine to save
```

Figure 1.

choice of using the space bar for single stepping or, alternatively, pressing the return key for fast tracing. It is possible to toggle between both of these modes, allowing the programmer to concentrate on a specific area of interest.

With the list trace window, the user has the option of seeing a count of the number of times a line has been executed. This information also can be displayed in histogram form.

In addition, variable windows display the sorted list of variables, their types, and current contents. The cursor/page control keys can be used to scroll up or down through a large number of variables.

The array window feature is similar to the variable window and is used to display arrays in the same fashion as single variable. Cursor/page control will also assist in performing slow, medium, and fast scrolls.

FOR/NEXT windows allow you to see the current status of your loops. This window will also display the active and pending loops.

DATA windows are also a very interesting feature. Displaying the BASIC line code with the *DATA* statement, items are depicted in reverse video once read. This enables a difficult tracing problem to be overcome.

File I/O windows supply the user with information regarding the buffers involved, the file name, and the I/O mode. They also display the BASIC I/O related statements as well as the data transferred. Figure 1 shows the file I/O window while the sample program was running.

Time trace windows show the program lines, and each is followed by the variable updated and its current value.

Subroutine windows show the active and pending subroutines. This is very effective in tracing nested subroutine calls.

Print windows will display information inputed and outputed by BASIC commands (e.g., *INPUT*, *PRINT*) to the console. Print/List will show a split screen, allowing the user to trace program line execution with console I/O.

Memory display windows show the contents in hexadecimal codes. The right portion of the screen displays the ASCII code equivalent, allowing the user to spot alphanumeric messages.

Pseudocode windows will display the BASIC command line executed and the equivalent pseudocode. While single stepping through the p-code, the user can watch the alternating pseudocode registers.

The windows mentioned are out of a selection of 26 available, and each and every one displays a status line at the very top.

Tracing with the windows can be done for single stepping, using the space bar, or in the fast-trace mode. During that time the user can choose the window of interest.

The screen can also show two windows at a time. This is extremely useful in having one side as the list-trace window and the other as data, loop, or I/O window. Changing windows is also possible with the split screen.

I would like to see the following changes and additions made on the product:

- Solve the flicker and snow screen problem during fast tracing
- Allow electronic drives to co-exist with Professional BASIC. Their presence will cause the interpreter to refuse any program line from being typed or read
- Implement some, more, or all of the PC-BASIC commands that were left out
- Create multi-line functions
- Develop modular libraries
- Re-direct dumping the screen to a text file
- Create search-and-replace capability

Professional BASIC is a well thought out and executed product. It addresses an audience of IBM PC BASIC programmers who need a very powerful tracing and debugging version of the language. It is loaded with features that allow you to perform tracing and debugging, leaving nothing to mystery.

By Namir Clement Shammas

Turbo Pascal v2.0

Hardware Requirements: Any Z80- or 8086-based machine running either MS-DOS, PC-DOS, CP/M-86, or CP/M-80
Price: $49.95
Available from: Borland International, 4113 Scotts Valley Drive, Scotts Valley, Calif. 95066, (800) 227-2400 ext. 968, in California: (800) 772-2666 ext. 968
Support: $89.95 for 8087 support, $29.95 to upgrade v1.0 to v2.0, $16.95 to upgrade from v1.0 to Turbo 87

When I first saw the ads for Turbo Pascal, I was quick to dismiss the product on account of the name. Seems like everything is "Turbo" these days—from bluejeans to yogurt—whether it employs exhaust-driven intake compression or not. But after months of using MT-Plus Pascal and doing 20-minute compiles and links between finding errors, I was ready to try something else. Like many other people, I tried JRT Pascal—after all, it was only 30 bucks!

I suppose many people are holding back on Turbo because the JRT experience has soured them on low-cost software. If so, hold back no longer.

Turbo Pascal is the Pascal to acquire. It has surprisingly few limitations, standard syntax with powerful extensions, and it's fast. Although the object code is only 10% to 20% percent faster than MT-Plus, it compiles instantly. You won't have time during compiles to read the manual—which, in a way, is too bad because the manual is quite readable.

Who can or cannot use Turbo Pascal? Every Pascal programmer using CP/M or MS-DOS should find it useful. It does have limitations, which I will describe. It also has many advantages. All in all, I feel the advantages outweigh the disadvantages.

Turbo Pascal is inexpensive—$50.00. You get an 8- or 5¼-in. diskette and a soft-cover bound manual about 5 by 8 in. in size.

Installing the system, however, is very difficult. The TINST program is very powerful because it allows you not only to configure screen escape codes but also to change editor commands (which default to WordStar-like commands). Unfortunately, it can be exasperatingly hard to operate. If you have one of the standard terminals, though, you won't have much trouble. Otherwise, watch out because the RESET button is the only way to start over if you make a mistake. If you have an IBM PC, take heart! It comes pre-installed.

Turbo Pascal's syntax is very standard. Programs that compile with MT-Plus or UCSD will usually compile unchanged with Turbo Pascal. Even so, *UNIT* is not supported, and tabs must be expanded to spaces in the source file.

Borland's new package acts as both an integrated editor and a compiler. When the compiler finds an error, it gives you an error number and prompts you to press ESC. When you do, you are put into the full screen editor at the point of the error. (The operation of the TAB key in the editor is a little weird).

For the most part, the editor emulates WordStar and is extremely fast and easy to use. You can (*E*)dit a source file, (*C*)ompile it into memory, and (*R*)un it, all in a matter of seconds. However, I'd recommend (*S*)aving the source file before you run it. If you don't remember the file name, ask for a (*D*)irectory. The letters in parentheses are the commands you give Turbo Pascal. It couldn't be much easier or faster than that.

With this Pascal compiler, you can compile into memory or produce a .COM file. There is no linking step. For this reason, there are no separate compilation or assembler subroutines. However, there is a standard *Include* facility for compiling long programs.

Turbo Pascal produces stand-alone .COM files. No run-time program is needed. The .COM files produced can be taken to another machine and run if the destination machine has as much or more TPA than the source machine or a switch in the (*O*)ptions command is set to a value to fit within the destination machine's TPA.

This compiler produces native code. This means programs tend to run very fast. It also means that if you code an endless loop you get an endless loop . . . ooops, reach for the reset switch!

Overlays can be created very easily in the revised version 2.0 of Turbo Pascal. If you have ever used overlays with Pascal MT-Plus, you probably avoid them like the plague. With this compiler, all you do to make a procedure into an overlay is put the word "overlay" on the front. That's all there is to it.

There are also some really good language extensions. Functions may return strings. Nested comments are allowed, and you can convert back into an enumerated type from an integer (inverse *ORD*). There is a random number generator, both for *REAL* values and for *INTEGER*s. Also, screen handling routines for cursor positioning and other screen functions are built-in. There are methods of accessing I/O ports and actual memory, and *BDOS* and *BIOS* functions are built into the language.

The package does have some some irritating features. Under CP/M, Turbo Pascal always returns to USER 0 on exit. If you don't have high/low intensity, or something like that, marked blocks are invisible in the editor. Strings passed to functions have to be declared as types with specific lengths unless *{$V-}* is used. All lengths of strings must match or you'll get a type mismatch. The editor repaints the screen much too often, and sometimes lines disappear that are still in the program. There is no *EXIT*—you have to use a *LABEL* and a *GOTO* to get out of a loop.

Table 1: Comparative benchmark timings based on the real matrix multiplication program presented in *BYTE*'s October 1982 issue.

	Turbo 1.0 (CP/M)	Turbo 2.0 (CP/M)	MT-Plus 5.5 (IBM)
Compile time (to memory):	2.5 sec	1.9 sec	n/a
(to disk):	4.1 sec	5.2 sec	54.3 sec
Link time:	n/a	n/a	33.6 sec
Run time:	29.5 sec	19.1 sec	21.5 sec
Total:	36.1 sec	26.2 sec	109.4 sec
Object file size:	10K	10K	16K

(CP/M benchmarks run on a 4 MHz Z80A system with 8-megabyte hard disk with no wait states. IBM benchmarks run on standard IBM PC XT with 256K RAM and a 10-megabyte hard disk).

Turbo Pascal is not copy protected. You may copy it onto your hard disk without any difficulty or onto floppies. If disk space is critical, most of the files provided are optional, and you can do without them.

Borland's documentation on this package is excellent. For some reason, every other Pascal compiler manual has an apology that goes something like, "This manual does not purport to be a reference on the Pascal programming language . . . " Turbo Pascal's manual is different. You can find things in it. You can even look things up in an index. Most amazing of all, the information in the book is actually understandable, complete, and useful. The book is softcover, bound, and 254 pages long. Version 2.0 comes with a 34-page booklet describing the added features.

For those of you who like to see benchmarks, I typed in Jerry Pournelle's real matrix multiplication benchmark out of the October 1982 *BYTE* and compiled the same source file in MT-Plus—using the *FPREALS* and *$Z* option—and Turbo (Table 1).

Of course, if you need to use Access Manager or high-speed assembly language device drivers, you may still have to use MT-Plus. However, most of the things that you needed assembly language for—such as I/O port or *BDOS/BIOS* calls—you will find implemented in Turbo Pascal. And, if you're still running an antique 8080 or 8085 processor, you can't use Turbo Pascal. It is only available for Z80- or 8086-family machines.

Perhaps the most amazing thing about the revised compiler is that IBM PC graphics and sound are now supported. You may select one of three different graphics modes. You may use black-and-white or color monitor. There are procedures to plot points and draw straight lines. (But arcs and circles would have been nice too). The sound function will generate a pure tone of specified frequency, or be quiet. These features will work on IBM PC hardware or perfectly compatible hardware only. I used an XT with an IBM-compatible monochrome graphics board, and the graphics did not work.

Windowing is now supported on the IBM PC with the revised version only, which now means that independent scrolling and cursor positioning on the screen are available. You may have only one active window; and once you've defined it, you've got to position the cursor to get into it. It necessarily follows that, to use multiple windows, you must keep track of the cursor position within each window yourself. In my opinion, this detracts from the automatic scrolling. The windows may not overlap.

Even still, the feature is very easy to use, and all the screen-handling functions may be used within a window. Cursor positions within the window are relative to the upper-left corner of the window. The display is very fast and clean, and graphics windows are also supported. Here, however, graphics coordinates remain as before; the windows simply look in on what would otherwise be a full screen.

8087 support is available in the updated version. For those of you who have the 8087 math coprocessor chip, this will really speed up calculations involving reals. However, the software does not check to see if you have an 8087. If you don't . . . reset button time again!

Turbo Pascal is recommended for anyone doing development work in Pascal. Its instant compilations and highly standard syntax make it a very useful tool even if the final version is processed with some other compiler. Turbo Pascal is also recommended for those of you who are looking for a way to learn Pascal—it is easy to use and well documented. Even if you decide you don't like Pascal, Turbo Pascal has a really fine screen editor that you can use for any language. Most of all, Turbo Pascal is recommended for anyone who wants to get the job done and doesn't care how little time it takes.

By Richard Rodman

ADVERTISER INDEX

	PAGE NO.	CIRCLE NO.
2500AD Software	40, 41	1
Awareco	26	2
BD Software	31	3
Borland International	1	4
Austin E. Bryant Consulting	66	5
CAPA	66	6
Carousel Microtools	49	7
Chromod Associates	60	8
The Code Works	72	9
CompuPro	Cover IV	10
Computalker	67	11
Computer Resources of Waimea	50	12
Creative Solutions	56	13
C Systems	12	14
C User's Group	68	15
C Ware	54	16
Datalight	67	17
Ecosoft	70	18
Eigenware Technologies	50	19
Foehn Consulting	68	20
Forth Interest Group	50	21
Allen Gelder Software	68	22
Greenleaf Software	50	23
Hawkeye Grafix	19	24
Introl Corp	62	25
JMI Software Consultants	48	26
Laboratory Microsystems Inc.	52	27
LA Software	68	28
Lattice, Inc.	76	29
Manx Software Systems	2	30
MBP Software & Systems Technology	55	31
Micro Focus	4, 5	32
MicroMotion	19	33
Miller Microcomputer Service	70	34
Mountain View Press	57	35
Next Generation Systems	54	36
John D. Owens Associates	64	*
Parsec Research Inc	72	37
Phoenix Software	43	38
Phoenix Software	45	39
Phoenix Software	47	40
Plum Hall	53	41
Poor Person Software	59	42
ProCode	52	43
The Programmer's Shop	58	44
Quest Research Inc.	6	45
Rational Systems, Inc.	63	46
RR Software, Inc.	Cover II	47
Sage Computer Technology	Cover III	48
SLR Systems	53	49
Softcraft, Inc.	24	50
Software Toolworks	73	51
Syntax Constructs Inc.	64	52
Systems Engineering Tools	73	53
Visual Age	17	54
Western Ware	68	55
Workman & Associates	59	56

THE INDEX ON THIS PAGE IS PROVIDED AS A SERVICE TO OUR READERS. THE PUBLISHER DOES NOT ASSUME ANY LIABILITY FOR ERRORS OR OMISSIONS.

Use the new . . .

COMPUTER LANGUAGE BULLETIN BOARD SYSTEM

300 / 1200 BAUD

(415) 957-9370

to telecommunicate with COMPUTER LANGUAGE.

By simply dialing this phone number with your computer and modem, you can now:

- SEND AN INSTANT "LETTER TO THE EDITOR".
- DOWNLOAD ANY PROGRAM LISTING OR MAGAZINE ARTICLE PUBLISHED IN THE MAGAZINE.
- SEND IN A MANUSCRIPT THAT YOU'D LIKE THE EDITOR TO CONSIDER FOR PUBLICATION IN COMPUTER LANGUAGE.
- UPLOAD ANY PROGRAM THAT YOU THINK THE READERS OF COMPUTER LANGUAGE SHOULD HAVE.
- REQUEST A SUBSCRIPTION TO THE MAGAZINE OR ADVERTISING INFO.
- PARTICIPATE IN AN INTERACTIVE ELECTRONIC MESSAGE SYSTEM WITH THE READERS OF COMPUTER LANGUAGE BY ASKING AND ANSWERING QUESTIONS POSED BY INDIVIDUAL READERS.
- COMMUNICATE WITH ANY OF THE COMPUTER LANGUAGE COLUMNISTS.
- AND MORE ....

NOTE: Once your modem has received a "connect", type several carraige returns to set the baud rate (either 300 or 1200). Most people have computers that do not require a "null" be set when this initial questions is asked.

Yet, if you prefer the U.S. Postal Service, send your correspondence to:

EDITOR
COMPUTER LANGUAGE
131 TOWNSEND ST.
SAN FRANCISCO, CA 94107

SUBSCRIBE TODAY!

Attn: Circulation Dept.

COMPUTER LANGUAGE

Subscribe to **COMPUTER LANGUAGE** at the Charter Subscription price today! Charter Subscription to **COMPUTER LANGUAGE** for only $19.95 — over 43% savings off the single copy price.

☐ Yes, start my Charter Subscription to COMPUTER LANGUAGE today. The cost is only $19.95 for 1 year (12 issues).

☐ I want to increase my savings even more — send me 2 years (24 issues) of COMPUTER LANGUAGE for only $34.95.

☐ Payment enclosed ☐ Bill me

Name __________

Company __________

Address __________

City, State, Zip __________

Offer expires 10/84. Please allow 6-8 weeks for delivery of first issue. Foreign orders must be prepaid in U.S. funds. Outside the U.S., add $12.00/year for surface mail or $30.00/year for airmail.

Guarantee: I can cancel my subscription at any time for a full refund.

SUBSCRIBE TODAY!

Attn: Circulation Dept.

COMPUTER LANGUAGE

Subscribe to **COMPUTER LANGUAGE** at the Charter Subscription price today! Charter Subscription to **COMPUTER LANGUAGE** for only $19.95 — over 43% savings off the single copy price.

☐ Yes, start my Charter Subscription to COMPUTER LANGUAGE today. The cost is only $19.95 for 1 year (12 issues).

☐ I want to increase my savings even more — send me 2 years (24 issues) of COMPUTER LANGUAGE for only $34.95.

☐ Payment enclosed ☐ Bill me

Name __________

Company __________

Address __________

City, State, Zip __________

Offer expires 10/84. Please allow 6-8 weeks for delivery of first issue. Foreign orders must be prepaid in U.S. funds. Outside the U.S., add $12.00/year for surface mail or $30.00/year for airmail.

Guarantee: I can cancel my subscription at any time for a full refund.

SUBSCRIBE TODAY!

Attn: Circulation Dept.

COMPUTER LANGUAGE

Subscribe to **COMPUTER LANGUAGE** at the Charter Subscription price today! Charter Subscription to **COMPUTER LANGUAGE** for only $19.95 — over 43% savings off the single copy price.

☐ Yes, start my Charter Subscription to COMPUTER LANGUAGE today. The cost is only $19.95 for 1 year (12 issues).

☐ I want to increase my savings even more — send me 2 years (24 issues) of COMPUTER LANGUAGE for only $34.95.

☐ Payment enclosed ☐ Bill me

Name __________

Company __________

Address __________

City, State, Zip __________

Offer expires 10/84. Please allow 6-8 weeks for delivery of first issue. Foreign orders must be prepaid in U.S. funds. Outside the U.S., add $12.00/year for surface mail or $30.00/year for airmail.

Guarantee: I can cancel my subscription at any time for a full refund.

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO. 22481 SAN FRANCISCO, CA USA

POSTAGE WILL BE PAID BY

COMPUTER
LANGUAGE

2443 FILLMORE STREET • SUITE 346
SAN FRANCISCO, CA 94115

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO. 22481 SAN FRANCISCO, CA USA

POSTAGE WILL BE PAID BY

COMPUTER
LANGUAGE

2443 FILLMORE STREET • SUITE 346
SAN FRANCISCO, CA 94115

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO. 22481 SAN FRANCISCO, CA USA

POSTAGE WILL BE PAID BY

COMPUTER
LANGUAGE

2443 FILLMORE STREET • SUITE 346
SAN FRANCISCO, CA 94115

FREE!

Free information from the advertisers of **COMPUTER LANGUAGE.**

1. Please fill in your name and address on the card (one person to a card).
2. Answer questions 1-3.
3. Circle the numbers that correspond to the advertisements you are interested in.

READER SERVICE CARD

Name

Company

Address

City, State, Zip

Country ____ Telephone number

Premier Issue. Not good if mailed after November 30, 1984.

Circle numbers for which you desire information.

1	11	21	31	41	51	61	71	81	91
2	12	22	32	42	52	62	72	82	92
3	13	23	33	43	53	63	73	83	93
4	14	24	34	44	54	64	74	84	94
5	15	25	35	45	55	65	75	85	95
6	16	26	36	46	56	66	76	86	96
7	17	27	37	47	57	67	77	87	97
8	18	28	38	48	58	68	78	88	98
9	19	29	39	49	59	69	79	89	99
10	20	30	40	50	60	70	80	90	100

Please complete these short questions:

1. I obtained this issue through:
   - ☐ Subscription
   - ☐ Computer Store
   - ☐ Retail outlet
   - ☐ Passed on by associate
   - ☐ Other
2. Job Title
3. The 5 languages that I am most interested in reading about (list in order of importance).

Comments

Attn: Reader Service Dept.

FREE!

Free information from the advertisers of **COMPUTER LANGUAGE.**

1. Please fill in your name and address on the card (one person to a card).
2. Answer questions 1-3.
3. Circle the numbers that correspond to the advertisements you are interested in.

READER SERVICE CARD

Name

Company

Address

City, State, Zip

Country ____ Telephone number

Premier Issue. Not good if mailed after November 30, 1984.

Circle numbers for which you desire information.

1	11	21	31	41	51	61	71	81	91
2	12	22	32	42	52	62	72	82	92
3	13	23	33	43	53	63	73	83	93
4	14	24	34	44	54	64	74	84	94
5	15	25	35	45	55	65	75	85	95
6	16	26	36	46	56	66	76	86	96
7	17	27	37	47	57	67	77	87	97
8	18	28	38	48	58	68	78	88	98
9	19	29	39	49	59	69	79	89	99
10	20	30	40	50	60	70	80	90	100

Please complete these short questions:

1. I obtained this issue through:
   - ☐ Subscription
   - ☐ Computer Store
   - ☐ Retail outlet
   - ☐ Passed on by associate
   - ☐ Other
2. Job Title
3. The 5 languages that I am most interested in reading about (list in order of importance).

Comments

Attn: Reader Service Dept.

Reader suggestions

Editorial Response Card

We want to hear your comments and suggestions about the premier issue of **COMPUTER LANGUAGE.** Your reader feedback will enable us to provide you with the information you want. Thank you for your help!

Comments:

☐ Yes, I have an idea for a manuscript:

☐ Yes, I'm interested in reviewing technical manuscripts.

☐ Yes, I'm interested in reviewing software.

Name:

Company:

Address:

City, State, Zip:

Phone Number:

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO. 22481 SAN FRANCISCO, CA USA

POSTAGE WILL BE PAID BY

COMPUTER LANGUAGE

2443 FILLMORE STREET • SUITE 346
SAN FRANCISCO, CA 94115

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO. 22481 SAN FRANCISCO, CA USA

POSTAGE WILL BE PAID BY

COMPUTER LANGUAGE

2443 FILLMORE STREET • SUITE 346
SAN FRANCISCO, CA 94115

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO. 22481 SAN FRANCISCO, CA USA

POSTAGE WILL BE PAID BY

COMPUTER LANGUAGE

2443 FILLMORE STREET • SUITE 346
SAN FRANCISCO, CA 94115

COMPUTER
LANGUAGE

Start your subscription to **COMPUTER LANGUAGE** at the Charter Subscription price of only $19.95. You'll receive the latest software design information every month at the lowest subscription rate we will ever offer.

☐ Yes, start my Charter Subscription to **COMPUTER LANGUAGE** for 1 year (12 issues). The cost is only $19.95 — over 43% savings under the single copy price.

☐ I want to increase my savings even more — send me 2 years (24 issues) of **COMPUTER LANGUAGE** for only $34.95.

☐ **Payment enclosed** ☐ **Bill me**

Name ____________________

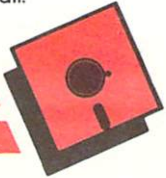
Company ____________________

Address ____________________

City, State, Zip ____________________

Offer expires 10/84. Please allow 6-8 weeks for delivery of first issue. Foreign orders must be prepaid in U.S. funds. Outside the U.S., add $12.00/year for surface mail or $30.00/year for airmail.

Guarantee: I can cancel my subscription at any time for a full refund.



NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO. 22481 SAN FRANCISCO, CA USA

POSTAGE WILL BE PAID BY

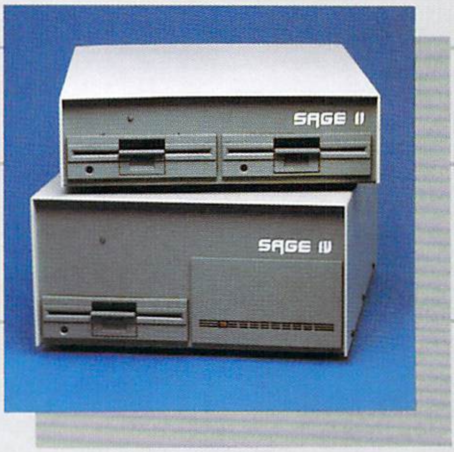
COMPUTER
LANGUAGE

2443 FILLMORE STREET • SUITE 346
SAN FRANCISCO, CA 94115

RFORMANCE

By Design

The recent Datapro Microcomputer User Survey reported a 3.8 overall user satisfaction rating out of a possible 4.0 for Sage Computers.



Sure, we like to read about ourselves scoring high marks in market studies. Our users do also. We appreciate the positive comments written about us by writers and editors around the world. But, as much as we enjoy the reports, it doesn't really surprise us.

We've designed performance into every computer system we manufacture. Not just speed, but flexibility, functionality and reliability. Sage has been building high performance 68000 multi-user systems longer than anyone, and we know that designing performance into our product requires time, attention to detail and a non-compromising attitude of doing things right.

Sage systems are available with nine different operating systems, 23 languages and over 300 application programs in 50 different categories. All systems come with a 90-day warranty, extendable to 3 years. And we have hundreds of dealers worldwide.

If you would like to know more about Sage and our Sage II and IV microcomputer systems, call or write today for your free copy of the 28-page Sage Product Catalog. It offers all you need to know about Sage, and how we design performance into every product we sell.

Reno: 702-322-6868
Dallas: 214-392-7070
Boston: 617-229-6868

SAGE
COMPUTER

Reno: 4905 Energy Way, Reno, Nevada 89502 (TWX 910-395-6073 SAGERNO)
Dallas: 14755 Preston Rd., Suite 600, Dallas, TX 75240
Boston: 15 New England Exec. Park, Suite 120, Burlington, MA 01803

CIRCLE 48 ON READER SERVICE CARD

HERE TODAY
HERE TOMORROW

When buying a computer, you can't limit yourself to just satisfying today's needs. **The best value in a system comes from its productivity . . . both for today and tomorrow.** CompuPro's System 816™ computer has that value. With all the power and capacity to handle your needs now and down the road.

System 816's longevity stems from top quality components . . . high storage capacity . . . the flexibility to handle a large variety of applications . . . and the speed to get the job done fast. Upgrading is easy, and when it's time to expand from single to multi-user operation, it's as simple as plugging in boards and adding terminals. Your system grows as you grow.

CompuPro also provides a library of the most popular software programs with your system and because it's CP/M® based, you have more than 3,000 other programs to choose from.

Even our warranty is for today and tomorrow. It spans 365 days — and includes the additional security of Xerox Americare™ on-site service nationwide for designated systems.*

What's more, CompuPro is one company you can count on to be around tomorrow. For more than ten years we've been setting industry standards, increasing productivity and solving problems.

For a free copy of our business computer buyer's primer, and the location of the Full Service CompuPro System Center nearest you, call **(415) 786-0909 ext. 206.**

CompuPro's System 816. The computer that's just as essential tomorrow as it is today.

CompuPro®
A *GODBOUT* COMPANY
3506 Breakwater Court, Hayward, CA 94545

*Available from Full Service CompuPro System Centers and participating retailers only.
System 816 and The Essential Computer are trademarks of CompuPro. CP/M is a registered trademark of Digital Research Inc. Americare is a trademark of Xerox Corporation.
System 816 front panel design shown is available from Full Service CompuPro System Centers only. ©1984 **CompuPro**

The Essential Computer™

CIRCLE 10 ON READER SERVICE CARD